Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicaciones

Trabajo Fin de Máster

Arquitecturas hardware avanzadas para el control de micro-redes eléctricas

Autor: Sergio de López Diz

Tutores: Emilio José Bueno Peña y Edel Díaz Llerena

2021

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería de Telecomunicaciones

Trabajo Fin de Máster

Arquitecturas hardware avanzadas para el control de micro-redes eléctricas

Autor: Sergio de López Diz

Directores: Emilio José Bueno Peña y Edel Díaz Llerena

Tribunal:

Presidente: Francisco Javier Rodríguez Sánchez

Vocal 1º: Ricardo Mallol Poyato

Vocal 2º: Emilio José Bueno Peña

Calificación: .....

Fecha: .....

"El presente es de ellos; el futuro, para lo que realmente trabajé, es mío."
Nikola Tesla

Agradecimientos

"De todas las maneras concebibles, la familia es un vínculo con nuestro pasado y el puente hacia nuestro futuro."

Alex Haley

En primer lugar, quiero agradecer a mi familia todo el apoyo, comprensión y ayuda que me han proporcionado en estos 2 años de estudios de máster. Os quiero Mamá, Papá, Alba y María.

También me gustaría dar las gracias a mi tutor Emilio, por todos los conocimientos que me ha transmitido y su disponibilidad total, a mi compañero y amigo Edel por su gran ayuda en el desarrollo de este trabajo y una especial mención a mi amigo Robert, trabajando y pasándolo bien juntos más de una década.

Por último, agradecer a todos mis compañeros del grupo de investigación por estar siempre dispuestos a echar una mano. Gracias a todos.

Resumen

El objetivo principal del presente trabajo es el diseño, implementación y evaluación de una plataforma de comunicaciones y de control para micro-redes eléctricas basado en un sistema Multi-Processor System-on-Chip (MPSoC). Los motivos fundamentales que justifican la utilización de estos dispositivos son la gran capacidad de cómputo, el alto grado de paralelismo y la flexibilidad para adaptarse a las múltiples topologías de convertidores de potencia presentes en las micro-redes actuales. La solución planteada se basa en el desarrollo de un sistema con una arquitectura Asymmetric Multi-Processing (AMP) que proporcione la heterogeneidad necesaria para el exigente control agregado de todos los elementos, así como la introducción del concepto de virtualización para la integración de una pila software que permita una abstracción *hardware* y un mecanismo de aislamiento entre las tareas críticas y no críticas en los diferentes niveles de control. La plataforma propuesta se ha evaluado sobre un convertidor real Back-to-Back (B2B) de 700 kVA actuando como un sistema de emulación de red para ensayos de certificación.

Palabras clave: MPSoC, AMP, micro-redes, virtualización.

Abstract

The main objective of this work is the design, implementation and evaluation of a communications and control platform for electrical microgrids based on Multi-Processor System-on-Chip (MPSoC). The fundamental reasons that justify the use of these devices are the high computing capacity, the high degree of parallelism and the flexibility to adapt to the multiple topologies of power electronics converters present in today's micro-grids. The proposed solution is based on the development of a system with an Asymmetric Multi-Processing (AMP) architecture that provides the necessary heterogeneity for the demanding aggregate control of all systems, as well as the introduction of the virtualization concept for the integration of a software stack that allow a *hardware* abstraction and an isolation mechanism between critical and non-critical tasks at different levels of control. The proposed platform has been evaluated on a real 700 kVA Back-to-Back (B2B) converter acting as a network emulation system for certification tests.

Keywords: MPSoC, AMP, microgrids, virtualization.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Lista de acrónimos

ADC	Analog-Digital-Converter.
AMP	Asymmetric Multi-Processing.
APU	Application Processing Unit.
B2B	Back-to-Back.
CMA	Contiguous Memory Allocator.
CSC	Control Secundario Centralizado.
DER	Recursos Energéticos Distribuidos.
DESC	Control Secundario Descentralizado.
DSC	Control Secundario Distribuido.
DSP	Digital Signal Processor.
DTB	Device Tree Blob.
DTC	Device Tree Compiler.
DTS	Device Tree Source.
ELF	Executable and Linkable Format.
EMS	Energy Management System.
FDoF	Full-Degree-of-Freedom.
FPGA	Field-Programmable Gate Array.
FQB	Frequency Reactive Power Boost.
FSBL	First Stage Boot Loader.
GaN	Nitruro de Galio.
GEI	Gases de Efecto Invernadero.
GPU	Graphics Processing Unit.
GPUs	Graphics Processing Units.
HTTP	HyperText Transfer Protocol.
IPC	Inter-Processor Communication.

IPI	Inter-Processor Interrupt.
IPs	Intellectual Properties.
LCOE	Levelized Cost of Energy.
MAS	Multi-Agent System.
MPC	Model Predictive Control.
MPSoC	Multi-Processor System-on-Chip.
P2P	Peer-to-Peer.
PERC	Passivated Emitter Rear Cell.
PI	Proporcional-Integral.
PID	Process ID.
PL	lógica programable.
PLC	Power Line Carrier.
PLL	Phase-Locked Loop.
PMU	Platform Management Unit.
PR	Proporcionales-Resonantes.
PS	sistema de procesamiento.
PV	paneles fotovoltaicos.
PWM	Pulse Width Modulation.
RPU	Real-Time Processing Unit.
S.O	Sistema Operativo.
SC	Secondary Control.
SCADA	Supervisory Control and Data Acquisition.
SiC	Carburo de Silicio.
SMMU	System Memory Management Unit.
SMP	Symmetric Multi-Processing.
SoC	System-on-Chip.
SPWM	Sinusoidal Pulse-Width Modulation.
SSBL	Second Stage Boot Loader.
STATCOM	STATic COMpensator.
TFM	Trabajo Fin de Máster.
THD	Total-Harmonic-Distortion.
UIO	UserSpace I/O.
VM	máquinas virtuales.
VOC	Virtual Oscillator Controllers.

VPD Voltage Active Power Droop.
VSC Voltage Source Converter.
VSM Virtual Synchronous Machines.

WBG semiconductores de banda prohibida ancha.

Capítulo 1

Introducción

1.1 Motivación

El sector energético está experimentando una transición hacia un futuro más inclusivo, seguro, rentable, bajo en carbono y sostenible en los últimos años, siendo el impulso de las energías renovables uno de los componentes fundamentales. El sector eléctrico lidera la transición energética en curso, impulsada por la rápida disminución de los costes de electricidad de origen renovable, particularmente para la generación eólica y solar. El objetivo de España, así como de los países pertenecientes a la Unión Europea, es convertirse en un país neutro en emisiones de carbono, reduciendo principalmente los Gases de Efecto Invernadero (GEI) con la finalidad de hacer frente el cambio climático.

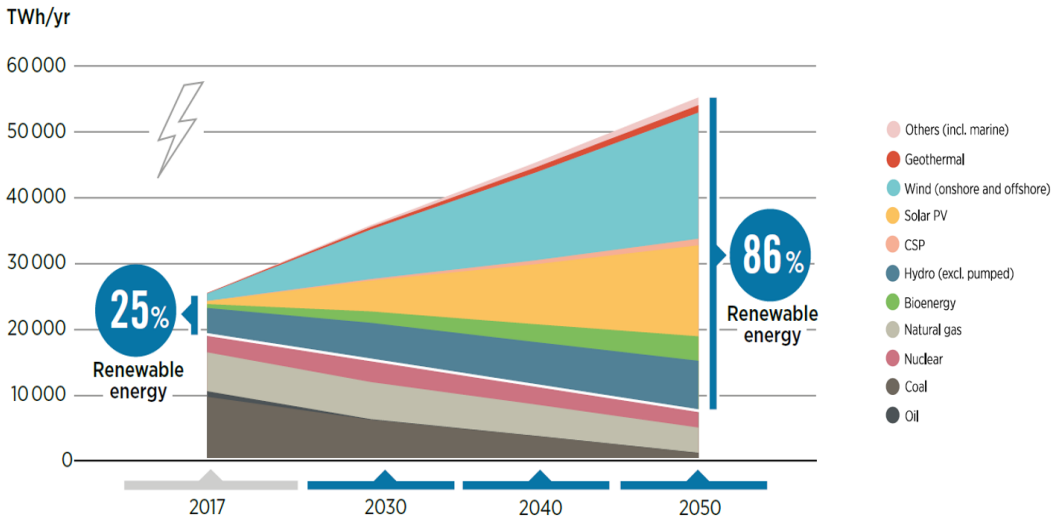


Figura 1.1: Estimación del crecimiento en el empleo de generación de energía de origen renovable. [1]

La generación de energía renovable se estima que aumente hasta aproximadamente un 40 % en 2030 y hasta un 86 % en 2050. Lo anterior, permite vislumbrar un futuro prometedor para la generación de energía renovable en los próximos años como se muestra en la figura 1.1, particularmente en tecnologías fotovoltaicas y eólicas.

Así mismo, se debe tener en cuenta, que el precio de los módulos solares fotovoltaicos (PV) se redujo en un 90 %, y el coste de la electricidad o el Levelized Cost of Energy (LCOE) de la energía solar fotovoltaica cayó un 77 % entre los años 2010 y 2020, cuyas expectativas en la próxima década es una

reducción considerable en los costes de estas tecnologías (figura 1.2).

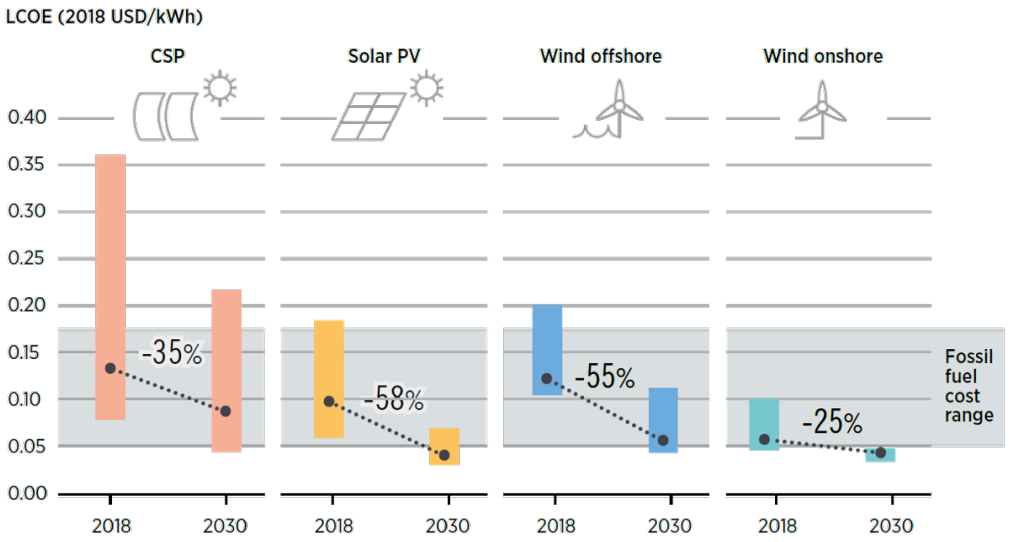


Figura 1.2: Estimación de la reducción en el coste de las tecnologías fotovoltaicas y eólicas hasta 2030. [1]

La disminución en el precio de la tecnología fotovoltaica es consecuencia de las últimas mejoras en términos de eficiencia por las nuevas y avanzadas arquitecturas empleadas en las células solares como la Passivated Emitter Rear Cell (PERC), consistente en la utilización de una capa dieléctrica reflectante para maximizar la radiación. Por otro lado, la energía eólica, tanto *onshore* (turbinas eólicas terrestres) como *offshore* (turbinas eólicas marinas), continúa su crecimiento en el uso de turbinas más grandes, que permiten cubrir un mayor área de barrido o *swept* aumentando su capacidad de generación de energía por un coste total de instalación en descenso.

El desarrollo de este nuevo paradigma se basa en la integración de nuevos modelos energéticos basados en la diversificación de las fuentes de energía, un mayor aprovechamiento de las energías renovables, conexión masiva de diferentes fuentes energéticas, de sistemas de almacenamiento, de vehículos eléctricos e híbridos... etc, así como la integración de nuevos actores en el sistema, como los prosumidores [18, 19], lo que dará como resultado redes descentralizadas más eficientes.

Todo esto es y será posible gracias al empleo de convertidores electrónicos de potencia que operan como interfaces de todos estos sistemas y la red eléctrica, proporcionando un flujo de potencia eficiente y estable.

La aplicación de estos sistemas de generación hace que se reduzca el uso de generadores convencionales, pero implican nuevos retos tecnológicos, pues las redes eléctricas están reduciendo considerablemente la inercia total equivalente, situación que será más evidente en los próximos años, si se cumplen los compromisos adquiridos, condicionando la estabilidad del sistema eléctrico completo.

La inercia total del sistema eléctrico proporciona el margen temporal suficiente para mantener la estabilidad en frecuencia, asegurando un equilibrio entre la generación y el consumo energético mediante el aumento o reducción de la energía producida ante el incremento o disminución de las cargas respectivamente. Si se comienza a reemplazar los generadores convencionales con fuentes de generación renovable que no poseen ningún tipo de inercia asociada puede provocar un aumento en la rapidez en el cambio de frecuencia en las situaciones de desequilibrio entre generación y carga, lo cual equivale a un sistema menos robusto.

1.1.1 Dispositivos semiconductores de potencia

Los dispositivos de conmutación basados en semiconductores son los elementos fundamentales que emplean los convertidores de potencia, cuyas características vienen determinadas por su aplicación final. En general, las características deseadas en cualquiera de ellos se detallan a continuación:

- **Características de encendido (ON)**: soportar una corriente elevada y proporcionar una caída muy pronunciada en la tensión.
- **Características de apagado (OFF)**: capacidad de bloqueo con una tensión alta e introducción de bajas pérdidas por corrientes de fugas.
- **Características de puerta**: tensión, corriente y potencia de actuación pequeñas.
- **Características de conmutación**: encendido (ON) y apagado (OFF) controlable a una frecuencia elevada, con unas transiciones en corriente y tensión muy pronunciadas y con bajas pérdidas en las transiciones.
- **Estabilidad térmica**: coeficiente de impedancia térmica pequeño en la unión interna-ambiente.

Las características descritas con anterioridad, entre muchas otras, representan aquellas propiedades idóneas para un dispositivo semiconductor. Sin embargo, no existen dispositivos ideales, por lo que la elección de las diferentes tecnologías presentes se basa en un equilibrio entre sus características y la aplicación concreta.

Los semiconductores basados en silicio son la opción preferida para su aplicación en convertidores de potencia, sin embargo, en la actualidad, se observa una penetración muy importante de los semiconductores de banda prohibida ancha (WBG) [20], Nitruro de Galio (GaN) y Carburo de Silicio (SiC) (figura 1.3).

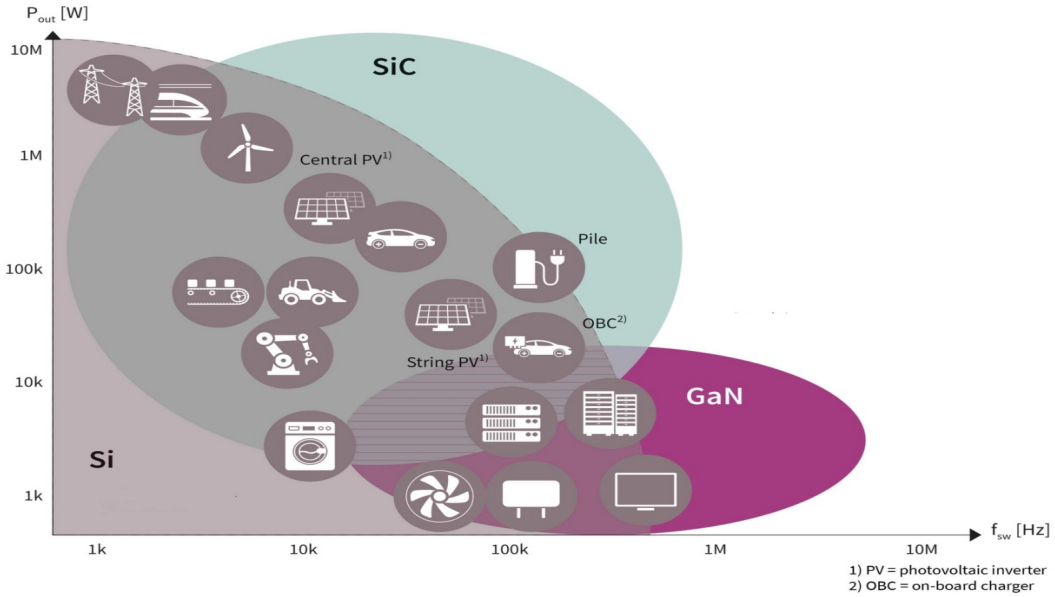


Figura 1.3: Clasificación de las nuevas tecnologías de semiconductores basada en la potencia máxima, frecuencia de conmutación y sus respectivas aplicaciones. [2]

Las principales ventajas asociadas a estas nuevas tecnologías son la posibilidad de trabajar a tensiones muy superiores a las de sus equivalentes de silicio, mejores características térmicas y el aumento en la frecuencia de conmutación.

Los transistores basados en SiC tienen una mayor capacidad de tensión (alrededor de los 1500V-1700V), bajas pérdidas de conmutación y permiten alcanzar altas frecuencias de conmutación y de potencia. Por otro lado, los transistores basados en GaN han tardado más en conseguir cierta posición en el mercado. Sin embargo, estos últimos años, su penetración esta aumentando en las aplicaciones de media potencia ofreciendo frecuencias de conmutación muy superiores a los anteriores con un coste menor.

1.1.2 Microgrids

En la actualidad, las microredes eléctricas proporcionan una arquitectura flexible para desplegar Recursos Energéticos Distribuidos (DER) basados en el empleo de convertidores electrónicos de potencia que actúan como interfaces con el sistema eléctrico y que puedan satisfacer las necesidades de generación y consumo energético de distintas comunidades, desde las áreas metropolitanas hasta las rurales con una reducción en el coste de la electricidad y en las emisiones de GEI.

Los principales factores que impulsan el desarrollo y la implementación de microredes en lugares con una infraestructura de red ya existente se pueden clasificar en tres grandes categorías: seguridad energética, beneficios económicos e integración de energía limpia [21].

1. **Seguridad energética:** este concepto está relacionado con el aumento creciente en la preocupación por las interrupciones de suministro eléctrico provocadas por el cambio climático, las cuales se esperan que sean más frecuentes y más graves en los próximos años, siendo necesario asegurar la resiliencia de la red a este tipo de situaciones. En este contexto, las microredes pueden proporcionar la potencia necesaria demandada por las cargas utilizando los distintos generadores distribuidos y las fuentes de almacenamiento cuando la red principal sufre una interrupción. Así mismo, dado que las redes eléctricas funcionan cerca de su capacidad crítica (problema aparentemente menor en una parte pequeña del sistema) puede conducir a un efecto avalancha que interrumpa el servicio de la red eléctrica [22]. La implementación de microredes puede aliviar este problema al segmentar el sistema en unidades de menor tamaño que pueden operar de forma aislada y autónoma.

2. **Beneficio económico:** las microredes ofrecen múltiples mejoras de eficiencia incluyendo menores pérdidas en las líneas de transmisión, así como transformaciones AC/DC, DC/AC y DC/DC con reducidas pérdidas gracias al empleo de convertidores electrónicos de potencia con los dispositivos de conmutación de nueva generación y técnicas avanzadas de control que permiten mejorar la calidad de las señales eléctricas. El concepto de servicios auxiliares es fundamental en el desarrollo de una microred eléctrica inteligente trabajando en modo isla incluyendo servicios como la regulación de frecuencia, seguimiento de carga, *black start*, control de potencia y tensión, así como calidad de potencia o *power quality* (potencia reactiva y compensación de armónicos). Lo anterior, junto con la reducción de los costes asociados a la energía fotovoltaica y eólica, se estima un aumento considerable en la inversión en el desarrollo de este tipo de arquitecturas, tal y como se observa en la figura 1.4.

3. **Integración de energía limpia:** una de las principales debilidades de las energías limpias, como la energía solar y eólica, es su variabilidad y su imposibilidad de controlar la generación eléctrica, por ello, las microredes se diseñan teniendo en cuenta esta generación variable con la integración masiva de sistemas de almacenamiento para compensar de forma local el balance entre generación y demanda energética.

Por todas estas razones, los sistemas de control, comunicaciones y la algoritmia que se despliega en estas nuevas redes eléctricas son fundamentales para garantizar la calidad de energía, la estabilidad de la red eléctrica y la continuación de suministro tanto en modo isla como ante perturbaciones severas.

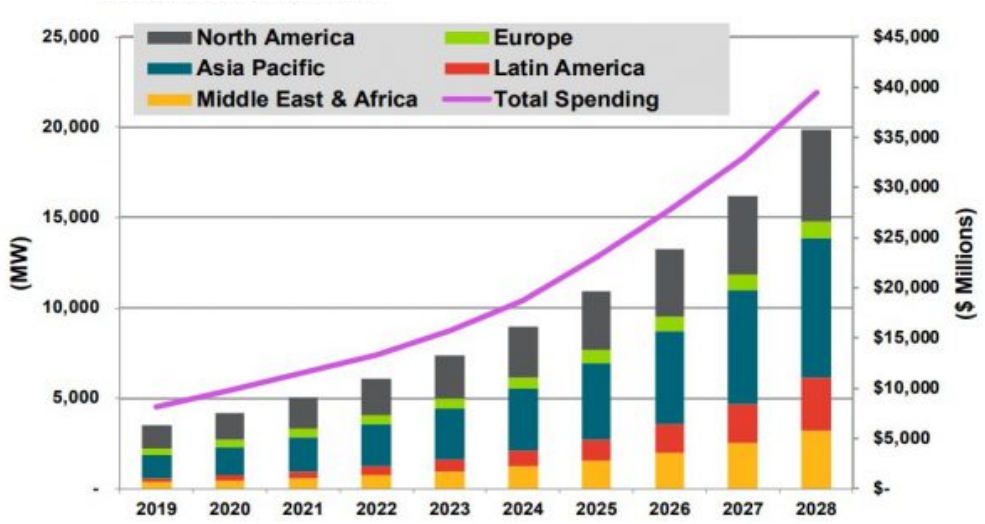


Figura 1.4: Previsión de inversión en el desarrollo de microredes. Fuente: *Navigant Research*

1.1.3 System on Chip

Hoy en día los System-on-Chip (SoC) son ampliamente utilizados por su potencia, flexibilidad y versatilidad para combinar una gran cantidad de elementos especializados en un único *chip*, permitiendo realizar estructuras cada vez más complejas en el diseño de componentes electrónicos. En estos dispositivos se integran núcleos de procesadores (con los elementos propios de un procesador como memorias caché, *timers*, controlador de interrupciones, unidades aritmético-lógicas...etc) junto con periféricos así como la interconexión entre todos ellos.

Un SoC permite reducir el consumo de energía, el coste y el espacio que pudiera ocupar un sistema de cómputo convencional, así como aportar una mayor potencia. Dependiendo del tipo de sistema, se puede enfocar para conseguir una transmisión de datos ultrarrápida para una gran variedad de funciones, incluyendo el procesamiento de señal, comunicaciones inalámbricas o inteligencia artificial. Otra posibilidad es diseñar el sistema con el objetivo de minimizar la latencia, integrando los distintos elementos de forma que se reduzca el retardo en las interconexiones de los componentes.

Un SoC+Field-Programmable Gate Array (FPGA) combina la potencia de una CPU junto con la flexibilidad que ofrece disponer de una FPGA para la implementación de lógica programable. Así mismo, incorporan una gran variedad de periféricos de entrada/salida, dependientes de la aplicación concreta para lo que ha sido diseñado el sistema. En la figura 1.5 se observan los componentes principales que pretende combinar esta nueva tendencia en el mercado tecnológico.

La evolución de estos dispositivos ha permitido la integración de un mayor número de unidades de procesamiento como microprocesadores o Graphics Processing Units (GPUs), desembocando en los denominados MPSoC. Este progreso supone un mayor rendimiento y velocidad a costa de aumentar de forma notable la complejidad en el diseño de las aplicaciones que se quieran desarrollar al tener múltiples subsistemas compartidos entre todas las unidades. Por otro lado, en el mercado de las FPGAs, valorado en 9000 millones de dolares en 2019, se espera un crecimiento anual del 9.7 % de 2020 a 2027, principalmente motivado por el sector de las telecomunicaciones y su empleo en aplicaciones como la conmutación y procesamiento de paquetes y la red de fibra óptica. Además, se espera que la llegada de la revolución de las redes de quinta generación (5G) impulsen aún más el desarrollo de este tipo de tecnologías por su capacidad de reconfiguración y conmutación de alta velocidad.

A pesar de que las aplicaciones de control de potencia, objetivo fundamental de este trabajo, no

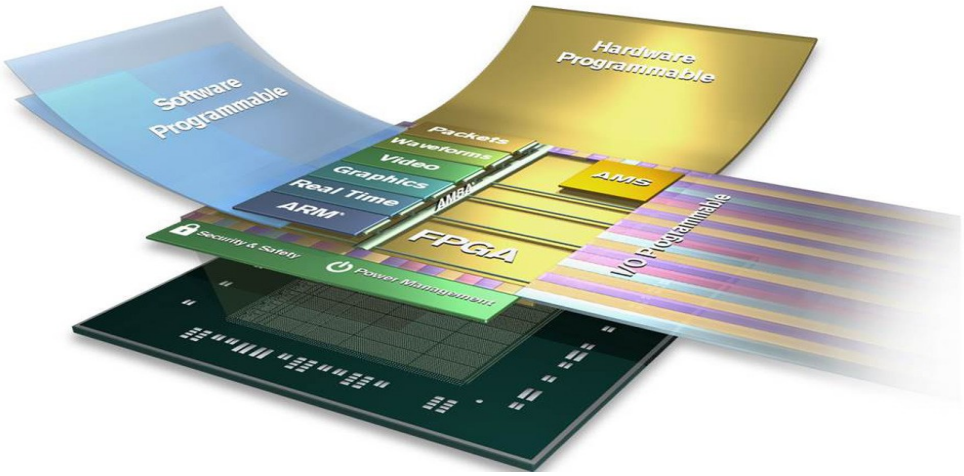


Figura 1.5: Ejemplo de combinación de componentes en un mismo dispositivo. Fuente: *www.xilinx.com*

constituye uno de los principales campos de aplicación de las FPGAs, se están beneficiando enormemente de todo el progreso realizado en otros campos, mucho más exigentes a nivel de cómputo que la primera. La consecuencia directa de lo anterior es que aún existe un gran margen de mejora para el control de sistemas de potencia utilizando los SoC, incorporando mejoras desarrolladas en otros sectores.

1.1.4 Virtualización

Hoy en día, la virtualización es una tecnología fundamental en los ordenadores de escritorio, permitiendo disponer de múltiples sistemas operativos en una misma máquina. Tradicionalmente, su empleo en sistemas embebidos no era posible por la necesidad de optimizar los SoC en términos de velocidad y recursos utilizados, además de que no se disponía de una plataforma *hardware* adecuada para manejar las capas *software* que proporcionan la abstracción necesaria para una correcta virtualización de los recursos. Las razones fundamentales que han permitido el desarrollo de la virtualización en los sistemas MPSoC se basan en las siguientes características de diseño [23]:

- **Optimización en la carga del sistema:** en un sistema de cómputo sin virtualización, se pueden utilizar múltiples sistemas operativos asignando de forma total o parcial los recursos *hardware* disponibles. Un ejemplo de ello es la integración del Sistema Operativo (S.O) *Linux* y *Windows* en una misma máquina. La problemática sucede al no poder trabajar con ambos S.O de forma simultánea sino que solo es posible trabajar con uno de ellos, quedando los recursos asignados al otro en desuso. La virtualización precisamente permite el manejo de múltiples instancias con diferentes S.O manteniendo todos los recursos de computación activos, distribuyendo y equilibrando la carga entre las máquinas virtuales (VM), proporcionando una mayor flexibilidad y modularidad.

- **Particionado y aislamiento:** el aislamiento y el particionado *software* son desafíos importantes en un entorno *multicore*. El planteamiento del problema es similar al descrito con anterioridad, incluyendo una restricción adicional: cada aplicación o *software* que se ejecute en cada uno de los núcleos no puede interferir con ninguna de las demás que se esté ejecutando simultáneamente. Las técnicas tradicionales están basadas en dotar de complejidad al *software* para garantizar que cada aplicación trabaje con los recursos finitos asignados. En contraposición, la virtualización funciona de forma radicalmente distinta, cada instancia tiene control total sobre los recursos disponibles en

su entorno virtualizado, es decir, utiliza los recursos sin saber si se está ejecutando otro *software*, aportando una mayor seguridad en la ejecución de múltiples aplicaciones. Lo anterior, permite reducir el tiempo de desarrollo de las aplicaciones al no tener ningún tipo de dependencia con la ejecución de otra instancia o código.

- **Escalabilidad y redundancia:** a medida que los sistemas multiprocesador han evolucionado en capacidad y velocidad, la necesidad de proporcionar redundancia y escalabilidad ha cobrado mayor importancia.

  Por una parte, el escalado pretende que la cantidad de *software* ejecutándose sea totalmente dependiente del nivel de demanda del sistema, es decir, si en cierto instante, la demanda de un servicio del sistema aumenta, se deben crear instancias idénticas del S.O que permitan responder con solvencia ante ese aumento de solicitudes. Una vez se haya respondido al pico de demanda, se pueden eliminar las instancias creadas previamente.

  Por otro lado, la redundancia impone que ciertas aplicaciones se encuentren siempre disponibles incluso cuando el sistema se encuentra cerca de su capacidad máxima. Lo anterior, se puede conseguir mediante la virtualización, a través de un monitor del sistema que compruebe de forma continua el estado de la aplicación considerada como imprescindible.

1.2 Sistema propuesto y objetivos

1.2.1 Sistema propuesto

En este apartado se presenta el sistema de control y comunicaciones propuesto para los convertidores de potencia de una microred eléctrica basado en un sistema multi-procesador. El esquema general de la arquitectura planteada para este Trabajo Fin de Máster (TFM) se muestra en la figura 1.6.

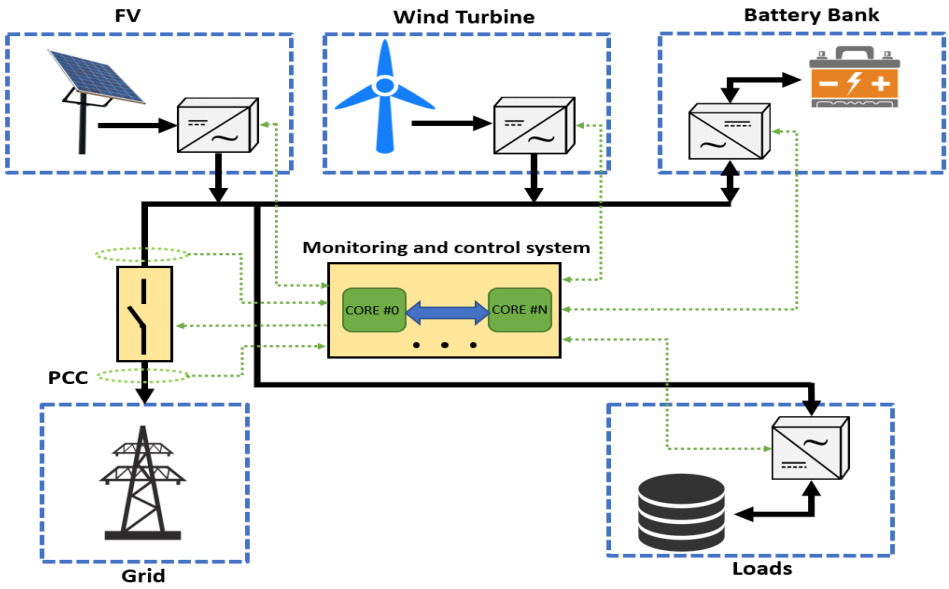


Figura 1.6: Esquema general del sistema propuesto para el control agregado de una micro-red eléctrica inteligente.

El elemento fundamental de control está formado por la última tecnología de MPSoC que dispone Xilinx, en concreto, el modelo *Zynq UltraScale+ MPSoC ZCU102*, el cual integra hasta cuatro núcleos

ARM-Cortex A53 de 64 bits destinados para la ejecución de tareas no críticas y 2 núcleos R5 de 32 bits, denominados de tiempo real, pues tienen asociados recursos de memoria propios para la implementación de tareas críticas y exigentes. Así mismo, incorpora una GPU ARM Mali-400 que permite obtener una capacidad de cómputo y paralelismo sumamente adecuada para aplicaciones que requieran cálculo matricial. Una descripción más detallada de las características de la placa de control empleada en este proyecto se encuentran en el capítulo 3.

La infraestructura del sistema se basa en la interacción conjunta entre los contextos de procesamiento que ofrece *Zynq UltraScale+*, entre la Application Processing Unit (APU), formada por los *cores* A-53, y la Real-Time Processing Unit (RPU), formada por el doble núcleo R5. Por una parte, la APU consta de la integración de un S.O de alto nivel, concretamente Linux, que permite la comunicación con el exterior de una forma sencilla y flexible, además de integrar los servicios de aplicación que permiten controlar y monitorizar el funcionamiento de cada convertidor. Los servicios implementados en este proyecto son un servidor web que permite visualizar y comandar el estado del convertidor, así como el envío de referencias y recogida de datos. Así mismo, el sistema dispone de una comunicación industrial como *MODBUS* con funciones similares a las comentadas para el servidor web. Por otro lado, los núcleos ARM-Cortex R5 se reservan para la ejecución de los algoritmos de control, dada su mayor exigencia computacional por los tiempos de muestreo de las señales a medir y la frecuencia de control para la conmutación de los dispositivos de potencia que forman los convertidores.

La comunicación entre ambos contextos se realiza mediante el empleo de librerías de *software* libre *OpenAMP*. En este TFM se han implementado las dos posibles configuraciones de comunicación *OpenAMP*, en el espacio del *kernel* del S.O o en espacio de usuario, comparando las diferencias entre ambas implementaciones y sus implicaciones.

Por último, se incorpora la posibilidad de utilizar la virtualización, mediante la inclusión de un hipervisor que permita la creación y control de VM que ejecuten otro tipo de S.O o código *baremetal* entre los distintos núcleos de la APU. Se debe tener en consideración que la comunicación entre la APU y la RPU con un hipervisor sólo es posible actualmente con una implementación en espacio de usuario.

En la figura 1.7 se muestra en mayor detalle la infraestructura del sistema propuesto, donde Linux junto con un hipervisor se ejecuta en los procesadores A-53, mientras que los algoritmos de control se destinan a los procesadores de tiempo real.

1.2.2 Objetivos

El objetivo de este proyecto es conseguir una plataforma *hardware/software* que permita controlar de forma agregada el mayor número de convertidores de potencia en el ámbito de las micro-redes eléctricas. Se trata de una temática novedosa que pretende servir de base para la gestión e implementación de controles de nivel primario y secundario. Para la consecución del objetivo general, es necesario que se cumplan los siguientes objetivos:

1. Revisión de las estrategias de comunicaciones y control empleadas en el control de micro-redes eléctricas.

2. Estudio y descripción de las funcionalidades que ofrece la placa de control basada en *Zynq UltraScale+*.

3. Estudio e implementación del protocolo de comunicaciones *OpenAMP*:

   (a) Comunicación con Linux en espacio del núcleo del S.O.

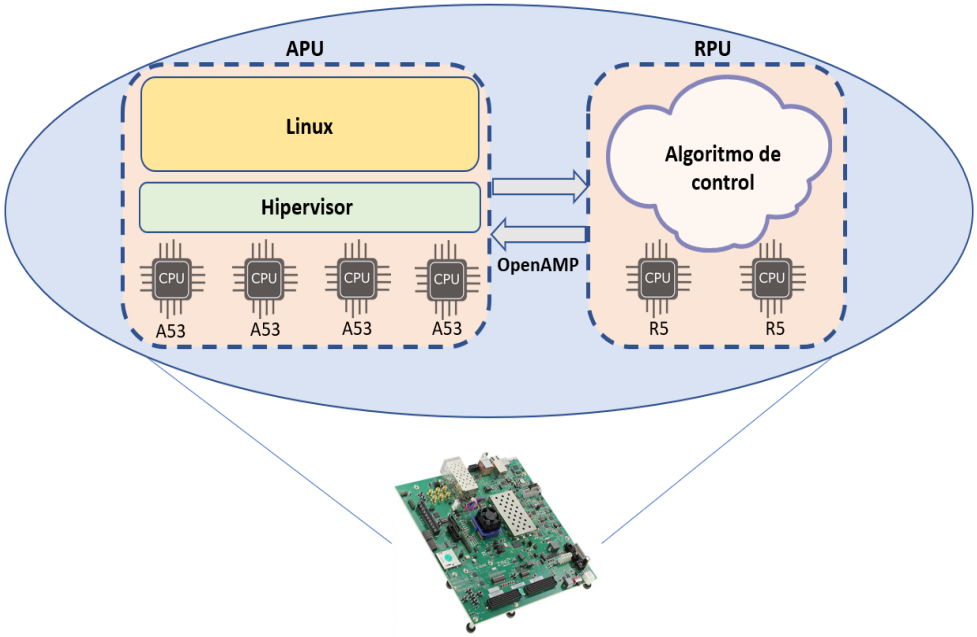


Figura 1.7: Detalle del sistema implementado.

(b) Comunicación con Linux en espacio de usuario.

(c) Ventajas y desventajas de ambas alternativas, así como mejoras para cumplir requisitos temporales más exigentes.

4. Implementación de los servicios de Linux, entre los que se incluye, servidor web y el protocolo *MODBUS* para intercambiar información con el exterior de la micro-red.

5. Proporcionar una arquitectura *software* flexible que permita cumplir los requerimientos temporales impuestos por el tiempo de muestreo utilizado en los controladores.

6. Empleo de la virtualización para segmentar la arquitectura conjunta que presenta por defecto el subsistema APU.

7. Validación de la plataforma implementada sobre un convertidor de potencia real.

1.3 Organización de la memoria

La organización de la memoria se expone a continuación:

– En el **Capítulo 2** se presenta una revisión al Estado del Arte actual en materia de arquitecturas de comunicaciones y control (primario y secundario) empleadas en las micro-redes eléctricas actuales, así como las implementaciones prácticas que se han llevado a cabo para la gestión de los convertidores de potencia.

– En el **Capítulo 3** se incluye un estudio teórico donde se exponen los conocimientos necesarios para la comprensión del desarrollo y la finalidad del sistema. Se abarca desde conceptos relacionados con la Electrónica de Potencia como las diferentes topologías, el modo de funcionamiento de las micro-redes y los principios de operación de los convertidores que actúan de interfaces entre todos los elementos, hasta una descripción detallada de la placa de control y los protocolos de comunicación utilizados

sobre ella. Finalmente, se presentan nociones básicas sobre la virtualización, los hipervisores y la creación y manejo de VM.

- El **Capítulo 4** contiene la explicación del principal objetivo del presente trabajo, donde se expone una descripción detallada de la implementación de la infraestructura introducida en el apartado 1.2.1. Primeramente, se expone el proceso de inclusión de un S.O sobre la plataforma y todos los componentes necesarios para proporcionar una comunicación adecuada entre APU y RPU. A continuación, se explica el procedimiento para ejecutar los algoritmos de control en los procesadores de tiempo real, en función de la alternativa elegida para realizar el canal de comunicación (espacio de usuario vs espacio de *kernel*). Posteriormente, se describen las propuestas de arquitectura de los servicios *software* implementados en Linux. Para finalizar, se comenta la integración de la virtualización en la plataforma y sus implicaciones.

- En el **Capítulo 5** se presentan los principales resultados experimentales obtenidos, donde se incluye la comparación en términos de rendimiento entre la alternativa de comunicación entre procesadores basada en espacio de usuario y en espacio de *kernel* del S.O, las ventajas e inconvenientes de las propuestas de infraestructura *software* y la validación de la plataforma desarrollada sobre un convertidor en un entorno real.

- El **Capítulo 6** finaliza con las conclusiones del presente trabajo y sus líneas de investigación futuras en el ámbito de la Electrónica de Potencia.

- En el **Capítulo 7 y 8** se presentan los requerimientos *hardware* y *software* necesarios, así como el presupuesto empleado para el desarrollo e implementación de este TFM.

Capítulo 2

Estado del Arte

2.1 Introducción

En este capítulo se incluye el estado del arte actual en el control de las micro-redes eléctricas inteligentes. Primeramente, se describen las principales alternativas de control que se pueden encontrar en la literatura, teniendo en consideración que las distintas técnicas aplicadas están fuertemente jerarquizadas, definiendo varios niveles en función de la velocidad de actuación requerida. El estudio está centrado en el control primario y secundario. A continuación, se realiza una revisión exhaustiva de los retos actuales y las futuras tendencias en el campo de las arquitecturas de comunicación, dado que los niveles más altos de control requieren una infraestructura de comunicaciones que permita la interacción entre los distintos nodos pertenecientes a la micro-red. Posteriormente, se detallan implementaciones prácticas de sistemas de control aplicados a convertidores electrónicos de potencia.

Finalmente, se exponen una serie de conclusiones en torno a los conocimientos descritos en este capítulo que permiten justificar la realización de este TFM y su aportación en el campo del control de las micro-redes del futuro.

2.2 Estructuras de control

Las micro-redes eléctricas o también denominadas como *microgrids*, forman un modelo de generación energética de baja-media tensión basado en la diversificación de las fuentes de energía de manera distribuida, permitiendo el intercambio de potencia desde y hacia la red eléctrica principal, así como la gestión y almacenamiento de energía. Una de las clasificaciones más sencillas de este tipo de sistemas es en función de la forma de onda de la señal de tensión con la que se trabaja: AC o DC.

El correcto funcionamiento y la coordinación entre los elementos del sistema requiere la aplicación de controles sobre los convertidores electrónicos de potencia que actúan con interfaces entre todos ellos.

Las estructuras de control en las micro-redes están basadas en un control jerárquico de tres niveles: primario, secundario y terciario. El control primario, también conocido como control local, se caracteriza por presentar la respuesta más rápida. Por otro lado, el control secundario es el encargado de la compensación de las pequeñas desviaciones que sufren los elementos del sistema conectados al bus principal y mejorar la compartición de energía entre ellos. Por último, como nivel más alto de la arquitectura se encuentra el control terciario, encargado de la gestión del intercambio de potencia, gestión de la energía o Energy Management System (EMS), así como la optimización del sistema y la planificación económica.

En la figura 2.1 se muestra un esquema de los diferentes niveles de control que se aplican en una micro-red AC, donde se observa con claridad que los niveles más bajos son aquellos con una mayor exigencia en cuanto a la velocidad de actuación, mientras que en los niveles más altos los algoritmos implementados aumentan su complejidad al tener en cuenta un mayor número de variables para lograr la optimización global del sistema.

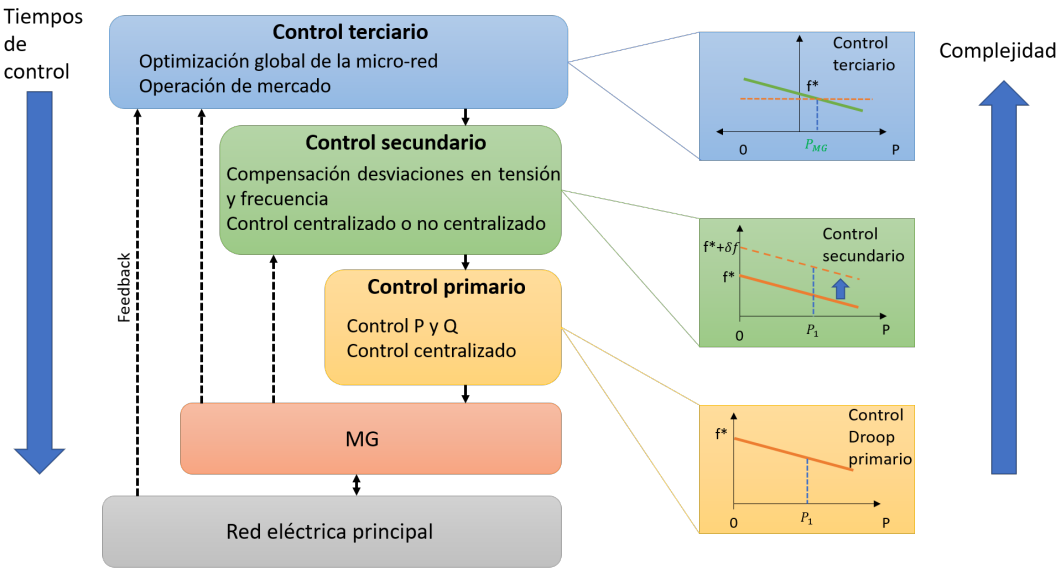


Figura 2.1: Esquema de la arquitectura jerárquica de una micro-red.

Las técnicas de control implementadas dependen fuertemente de la topología de la micro-red con la que se esté trabajando, si es DC o AC. En las siguientes secciones se expone el estado del arte en el control primario y secundario de ambas topologías.

2.2.1 Micro-red AC

Una micro-red AC está compuesta por generadores y cargas AC conectados entre sí a través de un bus AC que puede estar directamente conectado a la red eléctrica o trabajar en modo isla, es decir, sin conexión con la red y trabajando de forma autónoma. En este tipo de arquitecturas es fundamental el uso de convertidores electrónicos de potencia que actúen de interfaz entre los distintos elementos. En la figura 2.2 se observa un ejemplo de micro-red AC.

La exposición de las técnicas a nivel de control primario y secundario de los siguientes subapartados se han extraido de [4] y [24] respectivamente.

2.2.1.1 Control primario

El control primario trabaja directamente con las variables fundamentales del sistema, como son la tensión y la frecuencia del bus AC, asegurando el correcto seguimiento de las correspondientes referencias. Se debe tener en cuenta que no es necesario ningún tipo de comunicación, ya que el control se realiza de forma local, trabajando con las señales directamente medidas.

Los objetivos fundamentales que se enmarcan en el nivel primario son la regulación de la tensión y la frecuencia de las unidades de la micro-red, así como la detección del modo isla y el control de interconexión

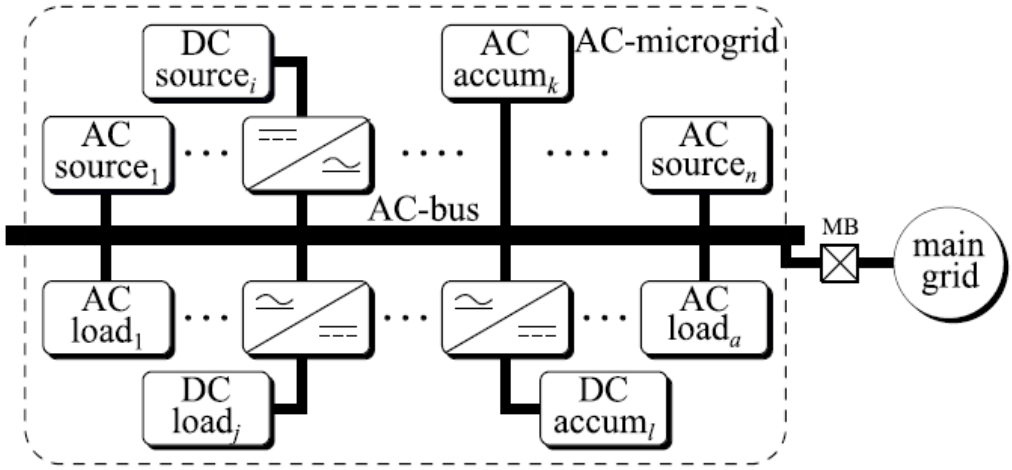


Figura 2.2: Ejemplo de topología micro-red AC. [3]

con la red principal. Se pueden diferenciar dos etapas: el control de las señales de salida del convertidor y el control en el intercambio de potencia.

Para la primera etapa se implementan distintas alternativas de controladores. Una de las más comunes es el empleo de un controlador Proporcional-Integral (PI) junto con una compensación *feed-forward* adicional que permite mejorar la respuesta dinámica y lograr una buena estabilidad del sistema incluso en presencia de cargas no lineales [25].

Otra posibilidad es el empleo de controladores Proporcionales-Resonantes (PR) para mejorar el rendimiento global del sistema con un controlador interno de corriente más robusto. Este tipo de controladores se pueden implementar utilizando la expresión 2.1:

$$C_{PR} = K_P + K_i \left(\frac{s}{s^2 + \omega^2} \right) \tag{2.1}$$

donde K_P representa la ganancia proporcional, K_i la ganancia integral del controlador y ω la frecuencia de resonancia. El objetivo de la aplicación de este tipo de controladores es conseguir un error nulo entre la señal y su referencia a través de una ganancia muy alta alrededor de la frecuencia de resonancia [26].

En la última década, los controles predictivos o Model Predictive Control (MPC) han adquirido una gran atención por parte de los investigadores. Este tipo de controladores están basados en la minimización del error entre la consigna y las predicciones de las variables de control en el siguiente tiempo de muestreo [27]. Una de las principales ventajas que ofrece MPC es la introducción de restricciones tanto en las señales de entrada como en las de salida en la realización de los cálculos de predicción.

La segunda etapa del control primario está centrada en asegurar el correcto intercambio de potencia entre las distintas unidades de generación distribuidas en la micro-red. Las principales técnicas de control están basadas en el *Droop*, cuyo objetivo fundamental es el control de la potencia activa y la regulación de la tensión de salida. En la literatura se pueden encontrar alternativas basadas en métodos convencionales [28] o no convencionales. Las técnicas convencionales asumen una alta impedancia inductiva en las líneas de conexión entre el convertidor de potencia y el bus, ajustando la referencia de tensión de salida y la potencia activa y reactiva. El control anterior está basado en la relación entre la potencia activa (P_o) y la frecuencia angular (ω_o) en función del valor de referencia de potencia (P^*) (expresión 2.2), así como la relación entre la potencia reactiva (Q_o) y la tensión de salida (V) en función de valor de referencia de potencia reactiva (Q^*) (expresión 2.3).

$$\omega_o = \omega^* - D_p(P_o - P^*) \tag{2.2}$$

$$V = V^* - D_Q(Q_o - Q^*) \tag{2.3}$$

La potencia aparente (S) que cada convertidor entrega al bus principal se calcula a través de la expresión 2.4:

$$S = V_{bus} I^* = \frac{V_{bus} E \angle \theta - \delta}{Z} - \frac{V_{bus}^2 \angle \theta}{Z} \tag{2.4}$$

donde $V_{bus}\angle\theta$ representa la tensión del bus, $E\angle\delta$, la tensión de salida del convertidor de potencia y $Z\angle\theta$ una impedancia genérica que modela la línea de conexión entre ambos. De la expresión anterior, se puede deducir la componente de potencia activa (P)(ecuación 2.5) y potencia aparente (Q) (ecuación 2.6) en función de la fase θ de la impedancia efectiva de la línea.

$$P = \frac{V_{bus} E}{Z} cos(\theta - \delta) - \frac{V_{bus}^2}{Z} cos(\theta) \tag{2.5}$$

$$Q = \frac{V_{bus} E}{Z} sin(\theta - \delta) - \frac{V_{bus}^2}{Z} sin(\theta) \tag{2.6}$$

Como se ha expuesto con anterioridad, las técnicas de control *Droop* convencionales asumen una alta impedancia inductiva, es decir, $\theta = 90^o$, sin embargo, esta asunción no se cumple en las micro-redes eléctricas, sistemas de transmisión de baja tensión cuya impedancia efectiva de línea equivalente se representa como una resistencia. En las expresiones 2.7 y 2.8 se muestran las ecuaciones de potencia activa y reactiva con la asunción de impedancia resistiva ($\theta = 0^o$) y un valor de δ lo suficientemente pequeño como para considerar $sin(\delta) \approx \delta$:

$$P \approx \frac{V_{bus} E - V_{bus}^2}{Z} \tag{2.7}$$

$$Q \approx \frac{V_{bus} E}{Z} \delta \tag{2.8}$$

La aplicación de un control *Droop* con líneas de transmisión modeladas como una resistencia [28] se denominan *Droop* de P/V o Voltage Active Power Droop (VPD) y *boost* de Q/f o Frequency Reactive Power Boost (FQB). Las características de VPD y FQB se muestran en la figura 2.3.

La propuesta anterior permite controlar el comportamiento de la micro-red, sin embargo, presenta una fuerte dependencia con los parámetros del sistema, por lo que no garantiza la correcta regulación de la tensión de salida ante cambios bruscos de carga o con cargas no lineales.

Para resolver la problemática del control ante la presencia de cargas no lineales, se emplea el método de transformación *virtual-frame* [29]. Se trata de una transformación ortogonal que permite desacoplar la potencia activa y reactiva, transformando sus valores reales en variables virtuales que se utilizan para aplicar las técnicas de control *Droop*. En la expresión 2.9 se ilustra la transformación de P y Q.

$$\begin{bmatrix} P_{virtual} \\ Q_{virtual} \end{bmatrix} = T_{PQ} \begin{bmatrix} P \\ Q \end{bmatrix} = \begin{bmatrix} sin\theta & -cos\theta \\ cos\theta & sin\theta \end{bmatrix} \begin{bmatrix} P \\ Q \end{bmatrix} \tag{2.9}$$

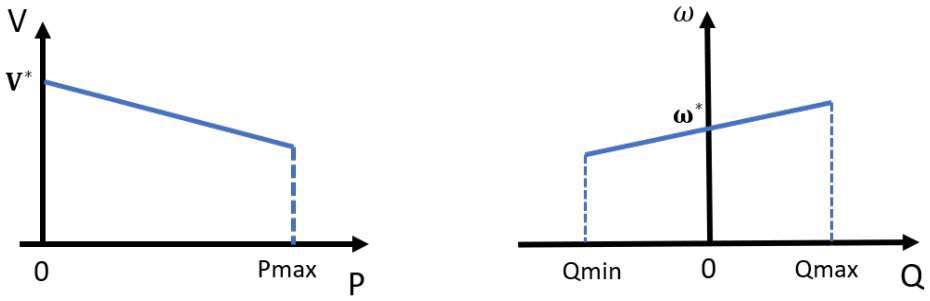


Figura 2.3: Características de las técnicas de control VPD y FQB aplicadas a micro-redes.

Otra alternativa presente en el estado del arte es el control de la impedancia de salida del convertidor ($Z_{virtual}(s)$). La referencia de tensión (v_{ref}) introducida al Voltage Source Converter (VSC) se obtiene a través de la expresión 2.10.

$$v_{ref} = v_o^* - Z_{virtual}(s) i_o \tag{2.10}$$

En la figura 2.4 se muestra el diagrama de bloques del control de la impedancia virtual de salida.

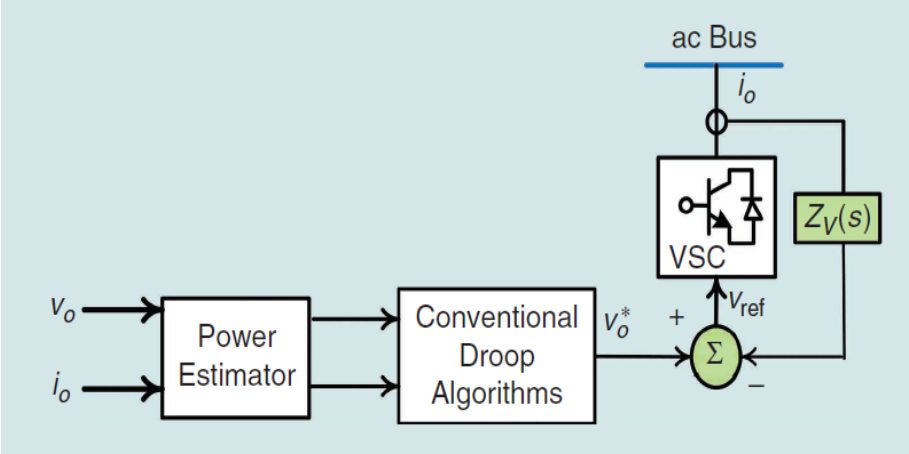


Figura 2.4: Diagrama de bloques del control de la impedancia virtual de salida [4].

2.2.1.2 Control secundario

El control secundario o Secondary Control (SC) se introduce para compensar las desviaciones o errores en régimen permanente de la tensión y la frecuencia en la aplicación del control *Droop* primario, así como mejorar el intercambio de potencia reactiva, reduciendo las corrientes circulantes y sus armónicos [30]. A partir de las ecuaciones 2.2 y 2.3, se puede expresar el objetivo del SC como se formula en 2.11 - 2.14.

$$\lim_{t \to t_f} \omega_i(t) = \omega^* \tag{2.11}$$

$$\lim_{t \to t_f} v_i(t) \approx v^* \tag{2.12}$$

$$\lim_{t\to\infty}(D_{P_i}P_i(t) - D_{P_j}P_j(t)) = 0 \tag{2.13}$$

$$\lim_{t\to\infty}(D_{Q_i}Q_i(t) - D_{Q_j}Q_j(t)) = 0 \tag{2.14}$$

Las ecuaciones anteriores representan la regulación de tensión y frecuencia para las $i, j = 1, ..., N$ unidades de generación de energía de la micro-red para un tiempo finito t_f, así como el intercambio de potencia entre todas ellas en régimen permanente. El signo $\approx$ en 2.12 indica que la tensión de salida de cada convertidor no sigue exactamente a la referencia, para conseguir la suficiente precisión en el intercambio de la potencia activa y reactiva, tal y como se detalla en [31]. La salida del control secundario (δu_ω y δu_v) se introduce como un término adicional en el control *Droop* primario (expresión 2.15 y 2.16).

$$\omega_o = \omega^* - D_p(P_o - P^*) + \delta u_\omega \tag{2.15}$$

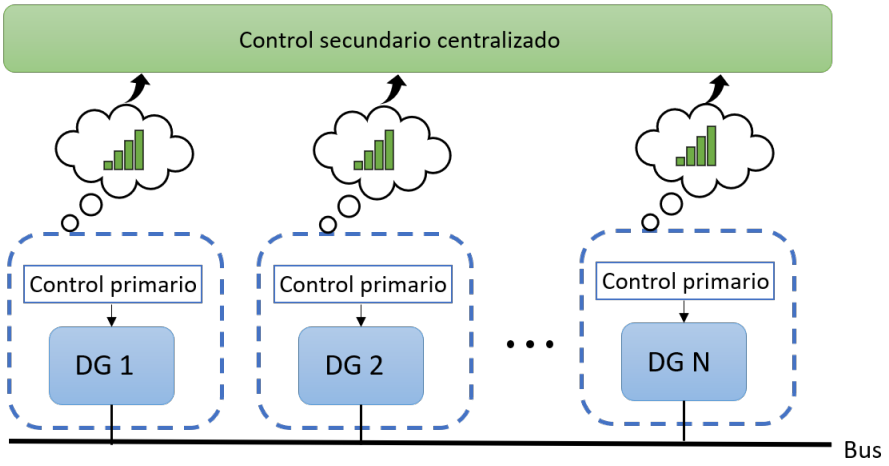
$$V = V^* - D_Q(Q_o - Q^*) + \delta u_v \tag{2.16}$$

La aplicación del SC se divide en función de la infraestructura de comunicación empleada, destacando tres categorías principales: Control Secundario Centralizado (CSC), Control Secundario Distribuido (DSC) y Control Secundario Descentralizado (DESC). En la figura 2.5 se presentan las arquitecturas de comunicación empleadas en la literatura para la implementación de los controles de segundo nivel.

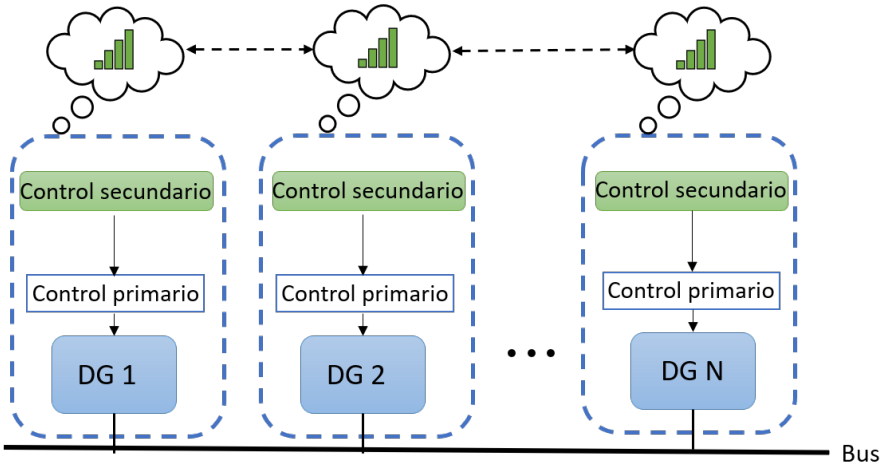
2.2.1.2.1 Control Secundario Centralizado En una estructura centralizada, un único controlador maneja y coordina las unidades de generación de energía, asegurando un correcto seguimiento de las referencias de tensión y frecuencia. Además, puede realizar la gestión de potencia activa y reactiva, así como la cancelación de armónicos. Cada unidad dispone de un canal de comunicaciones independiente con el controlador central, de forma que se envían las variables necesarias para implementar el algoritmo de control correspondiente. El rendimiento de la comunicación marca el comportamiento final de la micro-red, pues cualquier deficiencia en la transmisión de la información degrada la eficiencia global del sistema, pudiendo llegar a la inestabilidad de la micro-red ante un fallo del controlador central.

Los retos principales que se encuentran en el estado del arte con una estructura CSC son la cancelación de armónicos, la reducción de corrientes desbalanceadas y la mejora de la calidad y comportamiento global de la micro-red. Las técnicas de SC aplicadas a los retos mencionados con anterioridad, pueden basarse en el empleo de diferentes tipos de estructuras y controladores como en PI [5,32], funciones de coste [33,34] o métodos adaptativos basados en inteligencia artificial [35].

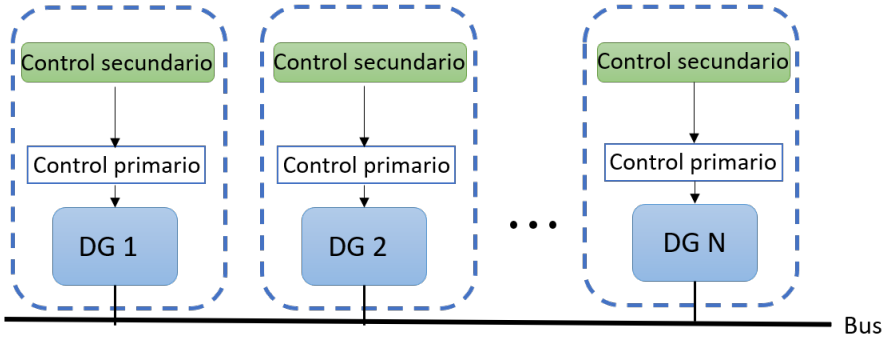
Un ejemplo en la compensación de armónicos en una estructura centralizada se muestra en la figura 2.6. En ella, se observa que el SC recibe como variables de entrada las secuencias positiva y negativa de la componente fundamental y los armónicos principales de las tensiones del bus AC en ejes dq. Los superíndices $+$, $-$, 1 y h representan la secuencia positiva, negativa, la componente fundamental y los distintos armónicos respectivamente. Previamente al envío de la información al controlador central, se filtran las señales anteriores mediante un filtro paso bajo de segundo orden. El SC se basa en el cálculo de los índices de distorsión de cada componente armónica (incluyendo a la componente fundamental de 50 Hz) y compararlo con sus correspondientes referencias, ajustando el error mediante controladores PI. La salida de cada controlador se multiplica con la entrada en ejes dq, obteniendo la compensación necesaria para los controles primarios de cada unidad. Se debe destacar el empleo de un bloque *deadband*



(a)



(b)



(c)

Figura 2.5: Arquitecturas principales control secundario: (a) CSC, (b) DSC y (c) DESC

para evitar el incremento en la distorsión en las circunstancias en las que los índices de distorsión sean menores a la propia referencia.

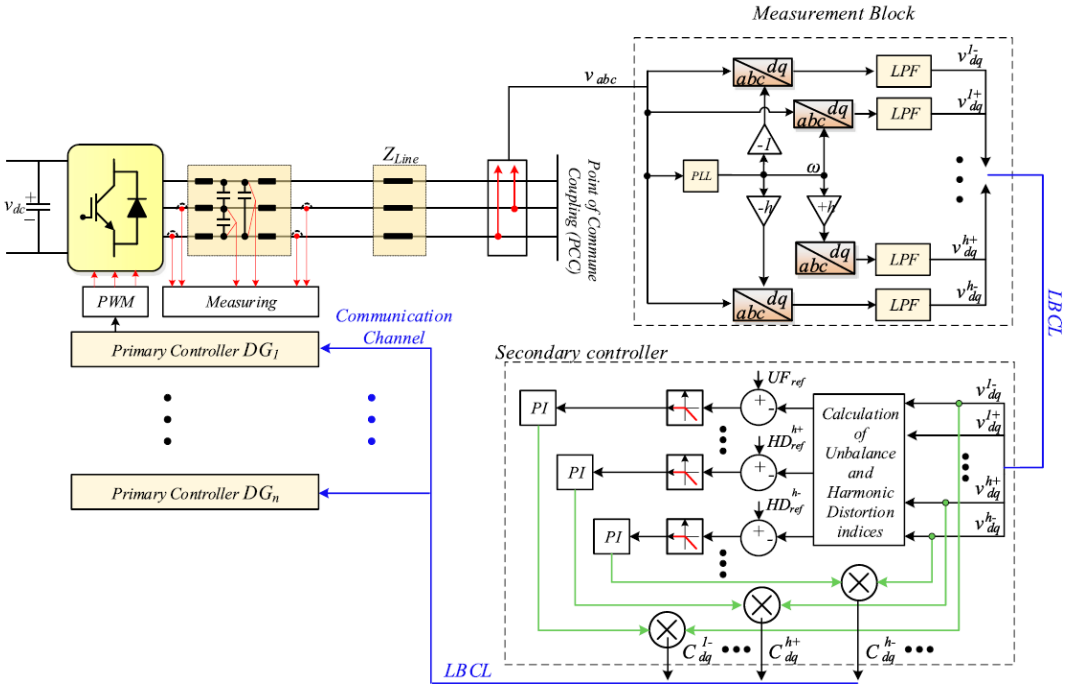


Figura 2.6: Esquema de un control secundario centralizado aplicado a la compensación de los armónicos presentes en las tensiones de salida [5].

2.2.1.2.2 Control Secundario Distribuido La distribución de las unidades de generación de energía de una micro-red es diversa y heterogénea, por lo que las propuestas de SC basadas en un control distribuido pueden contribuir en la mejora del funcionamiento global de la red así como la estabilidad de la misma. La finalidad de esta alternativa es la realización de un control de forma conjunta que cumpla un conjunto de objetivos marcados.

Las políticas de funcionamiento en el control distribuido se clasifican en tres tipos principales: *averaging*, consenso distribuido y basado en la detección de eventos.

El DSC con una estructura basada en el *averaging* consiste en la medición de la tensión y frecuencia de la señal de salida de cada unidad, enviando esta información a los demás convertidores electrónicos de potencia. De forma simplificada la señal de control se puede formular como se muestra en la expresión 2.17,

$$\delta u_i = K_i(s)(x^* - \frac{1}{N}\sum_{i=1}^{N} x_i(t)) \tag{2.17}$$

donde $x_i(t)$ representa la variable de interés y x^* su correspondiente referencia. En definitiva, se realiza el promedio de los valores recibidos de las N unidades que forman la micro-red.

La estrategia más común para la implementación del controlador $K_i(s)$ es el empleo de PIs [6, 36]. En la figura 2.7 se muestra un diagrama de bloques del control secundario basado en un PI, ajustando el control *Droop* primario del convertidor. En ella, se destaca en color azul el promedio de la tensión y frecuencia medida por los distintos elementos que forman la micro-red.

Por otro lado, el DSC basado en el consenso, consiste en la aplicación de algoritmos y protocolos que

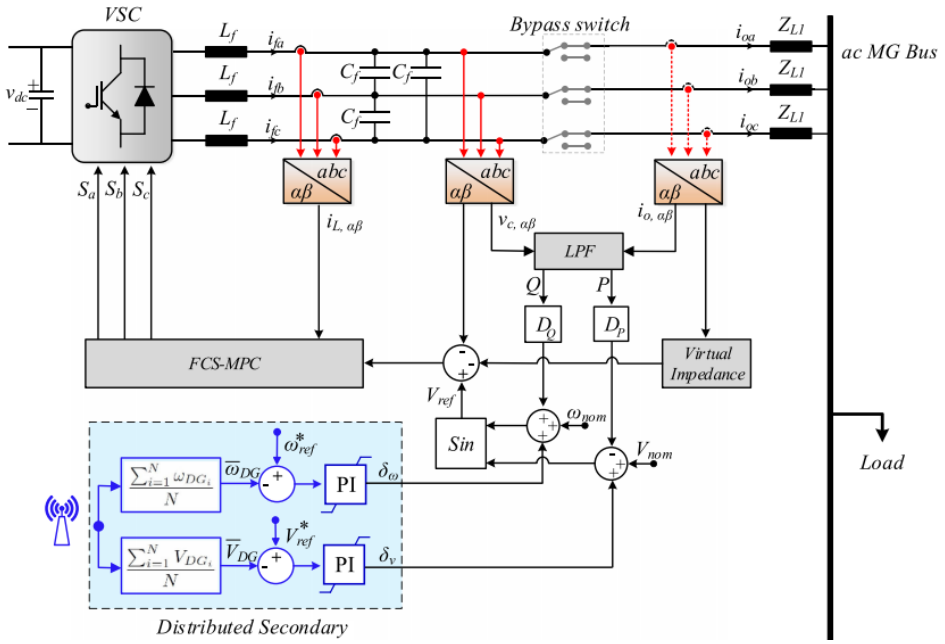


Figura 2.7: Esquema de un control secundario distribuido aplicado al mantenimiento de la tensión y frecuencia [6].

consiguen un funcionamiento conjunto de las distintas unidades. En la literatura se encuentran métodos basados en MPC [37], no lineales [38], adaptativos [39] y PIs [40].

Uno de los mayores retos en este tipo de estructuras es en qué unidades centrar el control, ya que en contraposición a un control de *following-leader*, es decir, en el que se designa en la micro-red un líder, el escenario actual es un intercambio dinámico en la selección de las unidades que actúan como *leaders* y *followers* para mejorar la robustez y la seguridad del sistema.

Por último, recientemente se está desarrollando estudios en el SC que permitan reducir el intercambio de información entre los distintos elementos. Las propuestas basadas en la detección de un determinado evento siguen la línea anterior, compartiendo información únicamente cuando una condición se cumple. En este contexto, se encuentran tres alternativas (figura 2.8) para activar este tipo de condiciones: activación por tiempo, por evento o auto-activación.

Una estructura con la activación de eventos basada en un temporizador (figura 2.8a) , se puede modelar como un muestreo en lazo abierto [41]. Por otro lado, la detección de eventos consiste en el envío de una señal de activación al SC cuando sucede un determinado evento, evitando la compartición de información entre las unidades de forma continua (figura 2.8b). Por último, el método *self-triggered* consiste en calcular una estimación de las señales de control del siguiente periodo, además de las señales del periodo actual, para realizar una monitorización del estado del sistema y enviar información cuando sea necesario (figura 2.8c).

2.2.1.2.3 Control Secundario Descentralizado Las propuestas de DESC no requieren una infraestructura de comunicaciones entre las distintas unidades para el control de tensión y frecuencia, ya que cada una actúa de forma independiente asegurando los valores nominales.

En la literatura se encuentran diferentes estrategias para la implementación de controles descentralizados. Una de las posibles alternativas es la introducción de filtros de *washout* en el control *Droop*. En [42] se introduce un modelo general simplificado de un filtro paso banda de *washout* aplicado a las

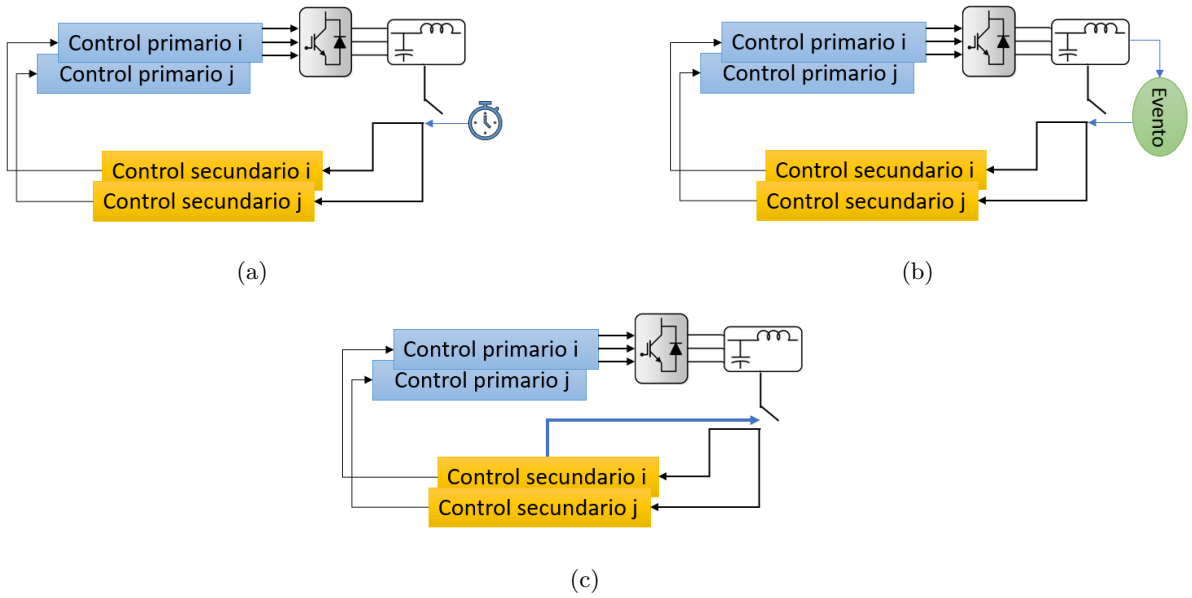


Figura 2.8: Propuestas control secundario basada en eventos: (a) Tiempo, (b) Detección de eventos y (c) *self-triggered*

micro-redes trabajando en modo isla:

$$\omega = \omega^* - \frac{D_P}{K_{p\omega}+1} \cdot \frac{\omega_c}{s+\omega_c} \cdot \frac{s}{s+\frac{k_{i\omega}}{K_{p\omega}+1}} \cdot p \tag{2.18}$$

$$v = v^* - \frac{D_Q}{K_{pv}+1} \cdot \frac{\omega_c}{s+\omega_c} \cdot \frac{s}{s+\frac{k_{iv}}{K_{pv}+1}} \cdot q \tag{2.19}$$

En las expresiones 2.18 y 2.19 se observa que el filtro paso banda generalizado se obtiene de la realización en cascada de un filtro paso bajo y un filtro paso alto. Un estudio de las implicaciones y efectos de un control totalmente descentralizado a través de este tipo de controles se describe en [43], incluyendo un análisis exhaustivo de rendimiento y robustez. De forma análoga, se pueden encontrar propuestas basadas en filtros de segundo orden [44].

Otra posibilidad para lograr un control descentralizado es mediante el empleo de las propias variables locales de cada convertidor, consiguiendo mitigar la latencia introducida por los protocolos de comunicaciones. En el estado del arte se describen diseños de reguladores lineales-cuadráticos para la regulación de la frecuencia [45, 46].

Finalmente, los métodos basados en la estimación de estados han cobrado una gran importancia en los últimos años, principalmente por eliminar su dependencia con la comunicación entre los elementos de la micro-red. Como es esperable, esta alternativa depende fuertemente del modelado que se realice del sistema y la calidad de la estimación de las variables. Los autores de [47] presentan un estimador de estados no lineal como prueba de DESC.

2.2.2 Micro-red DC

Actualmente, la mayoría de los sistemas eléctricos están dominados por el empleo de corriente AC, sin embargo, existe una clara tendencia por la implementación de sistemas DC en la transmisión y distribución de potencia motivado por el rápido desarrollo de convertidores electrónicos de potencia que trabajan en un elevado rango de tensiones de continua. En este contexto, una micro-red DC presenta una serie de ventajas comparada con una micro-red AC [48]:

1. Una gran parte de las energías renovables, como el empleo de paneles fotovoltaicos (PV) y pilas de combustible producen potencia DC. Así mismo, la integración de turbinas eólicas también es posible gracias al empleo de convertidores de potencia actuando de interfaz, con unas reducidas pérdidas en la conversión de energía.
2. No es necesaria la gestión de potencia reactiva ni sincronización en frecuencia.
3. La mayoría de los dispositivos de almacenamiento de energía son por naturaleza DC. De igual forma, las últimas tecnologías de baterías proporcionan un convertidor DC-DC interno que puede ser fácilmente integrado en el bus DC del sistema.

Un ejemplo de topología de micro-red DC se muestra en la figura 2.9.

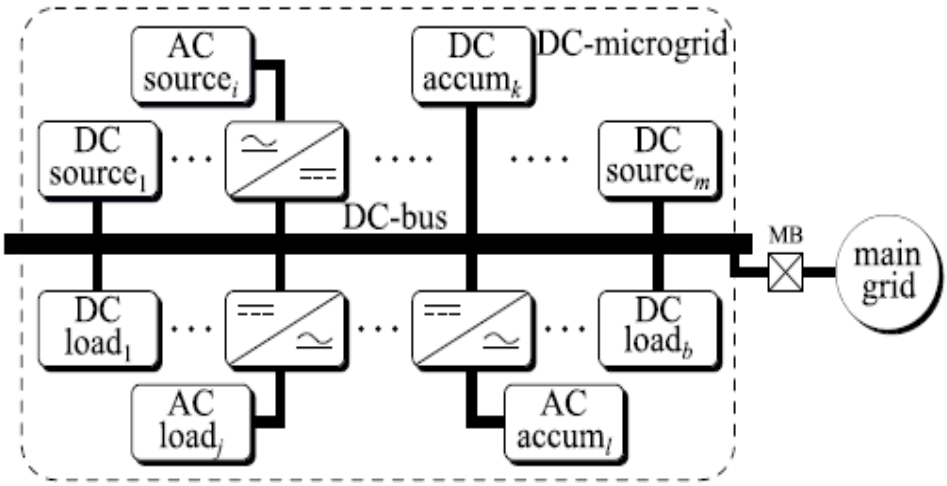


Figura 2.9: Ejemplo de topología micro-red DC. [3]

La exposición de las técnicas a nivel de control primario y secundario de los siguientes subapartados se han extraido de [7].

2.2.2.1 Control primario

Los convertidores electrónicos de potencia (tanto los convertidores AC/DC como DC/DC) son los elementos esenciales que permiten la integración de múltiples generadores, cargas y sistemas de almacenamiento de energía en un mismo sistema. Desde el punto de vista de control primario, las etapas que se encuentran en la literatura son equivalentes a las expuestas en el apartado 2.2.1.1, donde encontramos el control de lazo interno (regulación de tensión y corriente) y control *Droop* para el intercambio de potencia.

En el control interno se pueden diferenciar entre el aplicado a un inversor AC/DC y a un convertidor DC/DC. El primero de ellos (considerando un VSC de tres niveles), se basa en un control vectorial que

regula la corriente AC en ejes dq (figura 2.10). Tras una etapa de transformación de las corrientes en ejes dq, se ajustan su valor, mediante controladores lineales basados en PIs, en función de los valores de referencia I_d^* e I_q^*, cuyos valores se obtienen de la potencia activa y reactiva requerida.

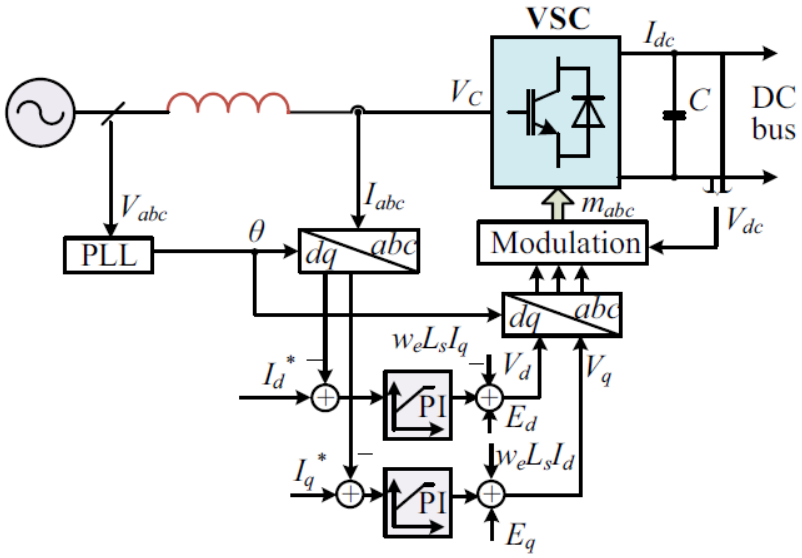


Figura 2.10: Control de corriente en un convertidor AC/DC trifásico. [7]

Por otro lado, el control aplicado a los convertidores DC/DC depende de su función específica y su topología, sin embargo, se puede generalizar en un modo de control de tensión y un modo de control de corriente. En definitiva, si el convertidor se basa en un control de tensión, se comporta como una fuente de tensión regulable, y si está basado en un control de corriente, opera como una fuente de corriente/potencia regulable. En la figura 2.11 se muestran los esquemas de control sobre un convertidor DC/DC, donde la señal de salida D representa el ciclo de trabajo del transistor y las referencias en tensión y corriente provienen del control *Droop* de la característica I-V que se expone más adelante.

Tal y como se ha descrito en la sección 2.2.1.1, la aplicación de un control *Droop* sobre la tensión ha sido ampliamente utilizado en el contexto de las micro-redes AC. Estos mismos conceptos se pueden aplicar a un sistema DC mediante la introducción de una "resistencia virtual" o también conocida como ganancia *Droop*.

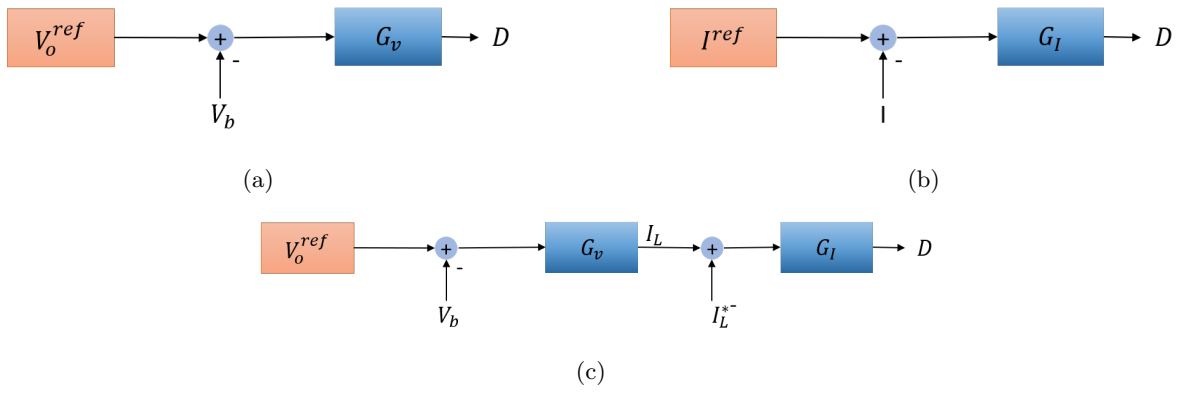
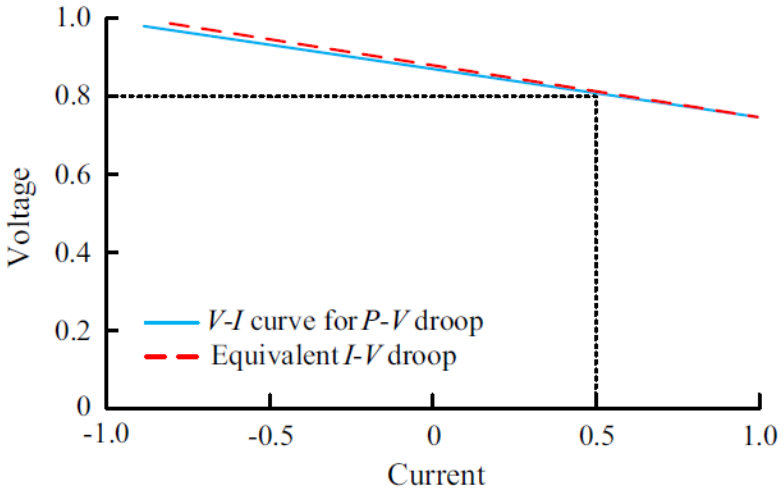


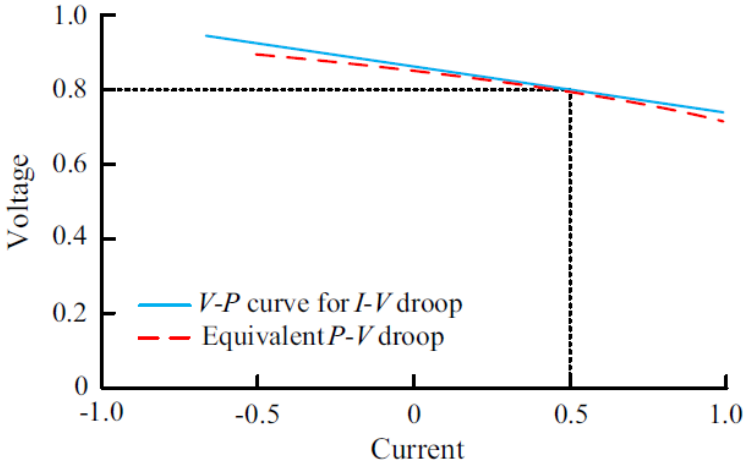
Figura 2.11: Tipos de controles de lazo interno en convertidores DC/DC: (a) Control de tensión, (b) Control de corriente y (c) Control tensión/corriente en cascada

En contraposición a las redes AC, no existe el concepto de potencia reactiva, sino que se define únicamente una relación entre la potencia activa P y la tensión V.

En la literatura se pueden encontrar diferentes implementaciones para el control *Droop* basado en la relación corriente/tensión (I/V) o en la potencia/tensión (P/V). En [49] se demuestra que los controles I-V y P-V son modelos prácticamente idénticos cuando se consideran pequeños errores en la tensión (figura 2.12).



(a) P/V Droop



(b) I/V Droop

Figura 2.12: Comparación control *Droop* de la tensión basado en potencia y corriente [7]

El control se puede plantear desde dos puntos de vista: modo de tensión y modo de corriente.

La propuesta basada en la tensión emplea una característica V-I que utiliza la corriente medida del bus para generar la referencia en tensión, tal y como se muestra en la expresión 2.20:

$$V_{dc}* = V_o - I_{dc}k \tag{2.20}$$

donde V_o representa la tensión nominal del bus, k la ganancia del control *Droop*, I_{dc} la corriente medida y V^*_{dc} la referencia de tensión DC calculada.

Por otra parte, la propuesta basada en la corriente utiliza la tensión medida del bus para calcular el

valor deseado de corriente. En la expresión 2.21 se presenta el cálculo necesario para obtener la referencia de corriente, donde V_{dc} es la medida de tensión y $I_{dc}*$ la referencia de corriente:

$$I_{dc}* = \frac{V_o - V_{dc}}{k} \tag{2.21}$$

El empleo de la técnica de control *Droop* en cada unidad de la micro-red define un método descentralizado que incrementa la modularidad y la fiabilidad del conjunto, sin embargo, la variabilidad de la impedancia de la línea de conexión con el bus DC, como podría ser por el cambio de temperatura, produce una degradación en el comportamiento de los elementos de la micro-red.

Las implicaciones de estos pequeños errores se pueden comprobar de forma sencilla suponiendo la conexión de dos fuentes de continua en paralelo. En la figura 2.13 se observa que si se emplea una ganancia de *Droop* pequeña, con una misma tensión de carga V_L, la diferencia en corriente entre ambas fuentes es $(I_{2o} - I_{1o})$, mientras que utilizando una ganancia más grande, la diferencia es $(I_2 - I_1)$, más reducida que la anterior, lo que permite concluir que ganancias del control *Droop* más elevadas tienen un mejor comportamiento en el reparto de corriente.

En contraposición, ante la condición de corriente en la carga I_L, la tensión del bus se reduce hasta V_{LA} con una ganancia pequeña y hasta V_{LB} con una ganancia elevada. Lo anterior, implica una mejor regulación de la tensión del bus trabajando con ganancias del *Droop* pequeñas.

Esta problemática que introducen los errores en el control es el reto que pretende solucionar el nivel de control secundario. Así mismo, se han investigado técnicas más sofisticadas para mejorar el rendimiento del sistema y plantear una solución al problema.

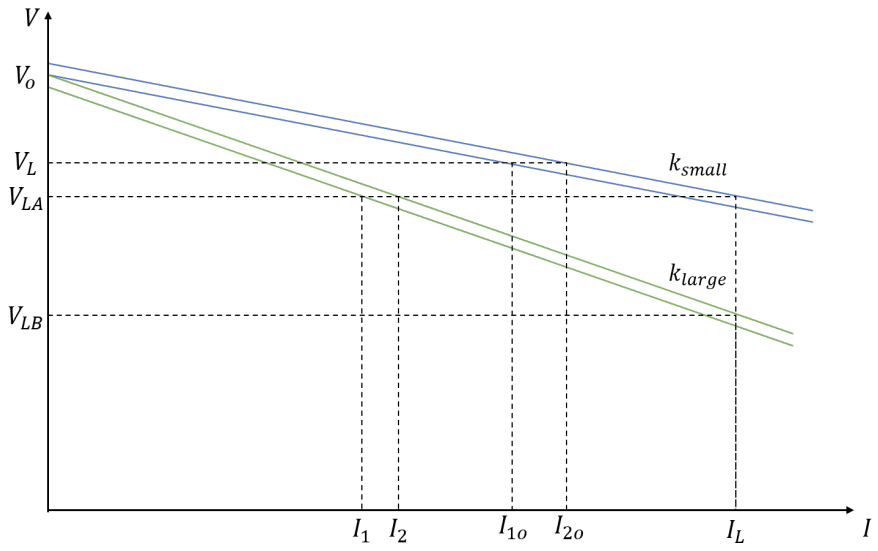


Figura 2.13: Ejemplo de las implicaciones del control *Droop* ante pequeños errores en las tensiones nominales.

En [50], se propone un control *Droop* inverso para el intercambio de potencia para convertidores DC/DC en paralelo, consiguiendo que la referencia de tensión de salida aumente a medida que la carga es más elevada.

Otra posibilidad es el empleo de técnicas de control no lineales, como en [51], donde se propone una modificación dinámica de la ganancia del Droop a medida que aumenta la carga, solventando el problema de los métodos lineales convencionales.

Finalmente, se pueden encontrar en la literatura alternativas de control adaptativas donde se realiza una planificación de las ganancias que se aplican en los controladores, ajustando un *Droop* más elevado para la regulación de tensión y uno más pequeño para una correcta compartición de la carga entre las unidades de generación. En [52] se consiguen raltivamente buenos resultados realizando una regulación de tensión e intercambio de potencia de forma simultánea.

2.2.2.2 Control secundario

En la sección 2.2.2.1 se ha introducido el reto que supone mantener en paralelo un intercambio de potencia adecuado y la tensión del bus estable en su valor nominal. El objetivo del control secundario es precisamente modificar la referencia de los controles primarios para solventar la problemática anterior. En la Figura 2.14 se muestra el principio de funcionamiento del control secundario. En ella, se observa en la implementación del control primario que el punto de operación se desplaza hasta el punto PO1 bajo la condición de corriente en la carga i_{dc1} y desde la tensión nominal v_o hasta el punto PO2 con una corriente en la carga i_{dc2}. El empleo del control secundario permite desplazar los puntos de operación a PO1' y PO2' manteniendo siempre el nivel de tensión nominal.

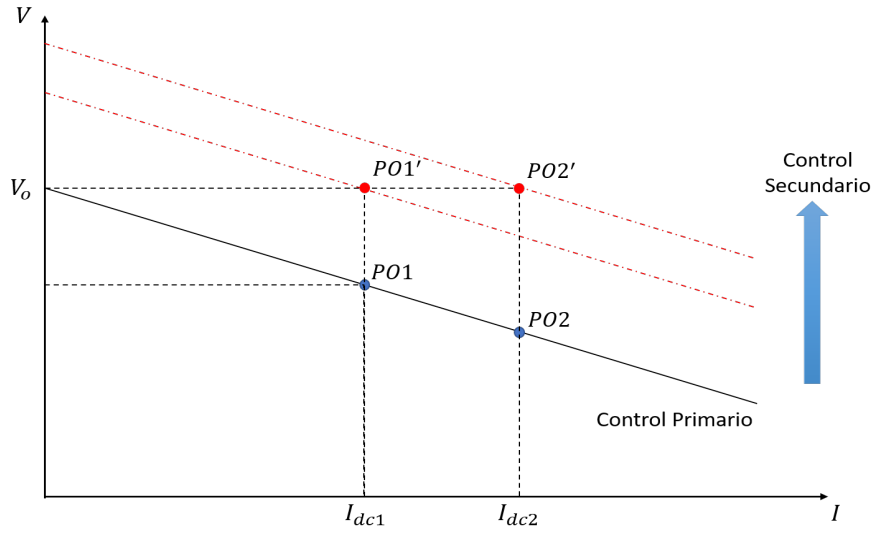


Figura 2.14: Principio de operación del control secundario.

En contraposición, al control descrito en las micro-redes AC (mantenimiento de tensión/frecuencia), la tarea principal del control secundario es eliminar las posibles desviaciones de la tensión de continua del bus así como mejorar aspectos relacionados con la calidad del intercambio de potencia. Nuevamente, desde el punto de vista del enlace de comunicación, se diferencian tres tipos de implementaciones para el control secundario: centralizado, distribuido y descentralizado.

En los siguientes apartados, únicamente se hace referencia a las técnicas de control de nivel secundario que se encuentran en la literatura utilizando los tres tipos de arquitectura de comunicación existentes, sin describir nuevamente su principio de funcionamiento, ya que los conceptos son iguales a los expuestos en 2.2.1.2.1, 2.2.1.2.2 y 2.2.1.2.3.

2.2.2.2.1 Control Secundario Centralizado La arquitectura CSC no es la más empleada en las micro-redes por el inconveniente de presentar un único punto de fallo, sin embargo, se encuentran alternativas para solventar los problemas de calidad de potencia y mantenimiento de la tensión nominal del bus. En [53] se propone un control clásico de tres niveles con buenos resultados en la regulación de

tensión y compartición de carga de forma simultánea. Por otro lado, en [54] se presenta un método de señalización del bus para micro-redes DC en modo isla.

2.2.2.2.2 Control Secundario Distribuido A diferencia del control centralizado, la información es intercambiada entre los distintos elementos que forman la micro-red, por lo que esta arquitectura es totalmente inmune al único punto de fallo. El control DSC se puede clasificar en tres categorías principales: esquema de intercambio de tensiones/corrientes medias, esquema de señalización del bus y esquema de control cooperativo.

En el contexto de un esquema de control basado en el intercambio de las tensiones y corrientes medias, en [8] se expone un control secundario basado en controladores PI para regular la tensión y corriente (figura 2.15).

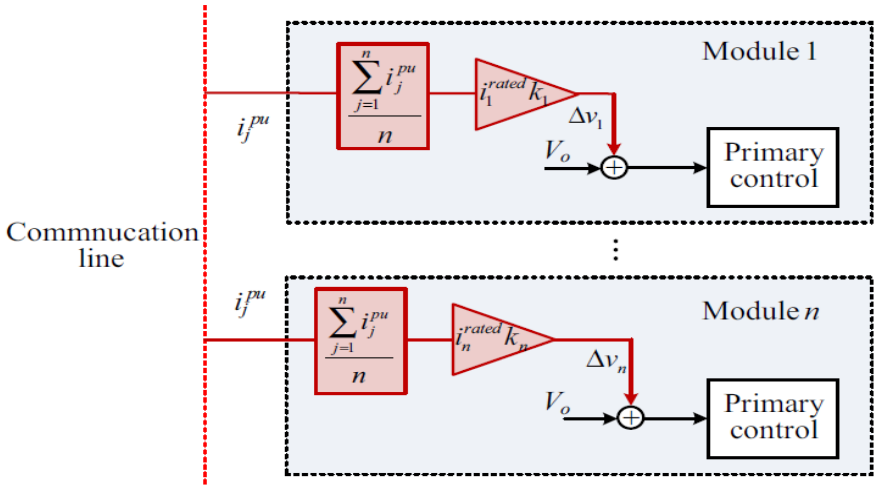


Figura 2.15: Esquema de control secundario propuesto en [8].

Otro trabajo relevante en el control distribuido es [55], donde se describe una implementación de control para las unidades de almacenamiento de energía, concretamente para baterías.

El empleo de la técnica de señalización del bus se encuentran en artículos como en [56] y [57]. En el primer trabajo, mediante un umbral asociado a la tensión del bus se supervisa la acción de los convertidores electrónicos de potencia. En el segundo, se presentan cuatro modos de operación de un sistema fotovoltaico modular.

Finalmente, el esquema de control cooperativo pretende que todos los nodos consigan alcanzar de forma conjunta un valor previamente consensuado. Para ello, se han presentado diferentes protocolos de control basados en algoritmos de consenso.

En [58] se presenta un modelo de micro-red DC empleando un algoritmo de consenso basado en tiempo discreto, analizando las implicaciones de la topología de comunicación. Por otro lado, en [59] proponen un sistema multi-agente para el control de dispositivos de almacenamiento de energía como baterías y supercondensadores, abordando el equilibrio del nivel de carga de los dispositivos y coordinando la distribución de energía.

2.2.2.2.3 Control Secundario Descentralizado En un control DESC, no es necesario el empleo de enlaces de comunicación entre unidades ni un controlador central, por lo que cada módulo actúa de forma independiente basándose en las medidas locales de cada convertidor.

En [60] se describe un control proporcional para reducir la influencia de la impedancia de los cables de conexión para mejorar el intercambio de potencia. Así mismo, [61] presenta un control basado en la inyección de una pequeña señal AC para integrar el concepto de control *Droop* utilizado en las micro-redes AC.

2.3 Arquitecturas de comunicaciones

Como se ha descrito en la sección 2.2, las estrategias de control necesitan redes de comunicaciones que permitan intercambiar información entre los distintos niveles de control y entre los distintos elementos que forman la micro-red.

Para asegurar un correcto envío de datos entre los nodos del sistema, deben definirse protocolos que regulen y definan el intercambio de información. Las redes de comunicaciones suelen emplear múltiples niveles o capas de protocolos basadas en el modelo OSI, convirtiendo los datos de una forma adecuada para su posterior transmisión.

En la actualidad, las instalaciones de micro-redes están basadas en arquitecturas centralizadas que implementan el esquema de control jerárquico y realizan la transmisión de información entre las distintas unidades. Un ejemplo de este tipo de arquitecturas centralizada que lleva varios años implementada es la denominada Supervisory Control and Data Acquisition (SCADA) [62]. SCADA utiliza una simplificación del modelo OSI basado en el empleo de únicamente 3 capas (capa física, capa de enlace y capa de aplicación), reduciendo los posibles servicios a ofrecer, pero que permite enlaces de comunicación directos sin necesidad de Internet. Algunos de los protocolos más populares en el contexto de l sector eléctrico son *MODBUS*, *PROFIBUS*, *CANBus* o *DNP3*, todos ellos basados en una infraestructura cliente-servidor.

Recientemente, se está produciendo un incremento en el interés de nuevas tecnologías de comunicación basadas en el uso de Internet, aprovechando la arquitectura TCP/IP que simplifica la comunicación entre diferentes entidades. Este cambio en los protocolos acompañan a los sistemas de control eléctrico, ofreciendo un mayor número de servicios. Nuevamente, se pueden mencionar la evolución de los protocolos expuestos con anterioridad, como *MODBUS/TCP*, *DNP3 over TCP* y *Profinet*.

A pesar de las mejoras que introducen los protocolos basados en TCP/IP, la principal limitación se encuentra en el modelo de control cliente-servidor cuyos inconvenientes se describen a continuación:

- Un fallo en el controlador maestro puede ocasionar múltiples fallos en el comportamiento normal del sistema, incluso dejar inoperativa toda la micro-red.
- En una estructura maestro-esclavo, de forma estricta, los nodos de la micro-red (esclavos) no pueden iniciar la comunicación con el maestro, lo que supone un problema ante ciertas situaciones, dificultando la gestión de la información.

Las desventajas comentadas pueden ocasionar un peor servicio, cuellos de botella e infra-utilización de los distintos recursos interconectados en la micro-red.

Otra tecnología alternativa para el envío/recepción de información es la denominada como Power Line Carrier (PLC). Este tipo de tecnología emplea las líneas de transmisión de energía como medio para el intercambio de información bidireccional, aprovechando la infraestructura ya instalada, por lo que, sin duda, es la alternativa con el menor coste efectivo. Sin embargo, presenta una serie de inconvenientes que afectan a la comunicación como el ruido, distorsiones, cambios en la impedancia y la frecuencia, así como la atenuación de la señal [63].

La evolución de las micro-redes inteligentes actuales está basado en la integración de un mayor número de recursos energéticos de forma distribuida. Por ello, la infraestructura de comunicaciones se está transformado también a un modelo distribuido que permita eliminar las desventajas de la arquitectura centralizada, mejorando la fiabilidad del sistema.

En este tipo de estructuras, se definen unas entidades denominadas agentes, sistemas de computo capaces de realizar tareas de forma autónoma y con la capacidad de comunicarse con los nodos vecinos para resolver problemas mediante la cooperación, coordinación, sincronización y negociación [64]. Como resultado, se pueden seguir planteando estructuras centralizadas implementadas sobre una arquitectura descentralizada.

2.3.1 Topologías de red

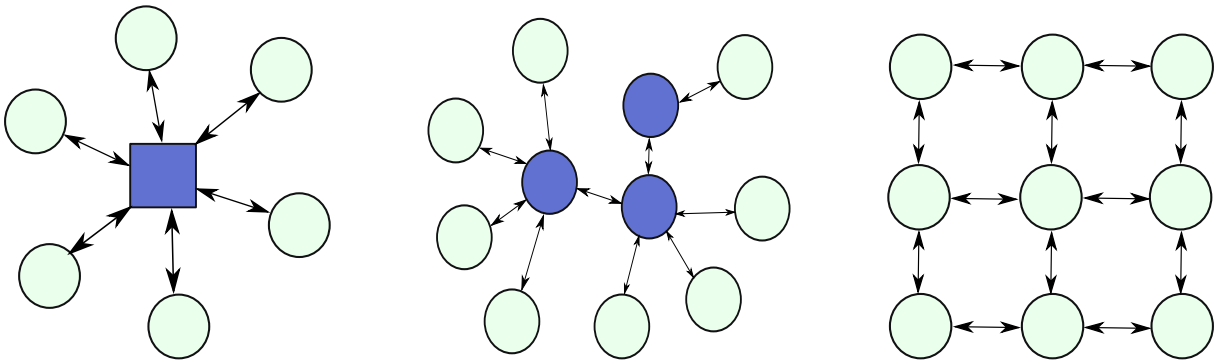
Las topologías de los nodos que forman la micro-red tienen una importancia crucial en la eficiencia y en el correcto funcionamiento del sistema completo. En función del grado de descentralización, las topologías de red se pueden clasificar en las siguientes arquitecturas [65]:

1. **Centralizada:** En esta topología se emplea un nodo central que actúa de servidor almacenando la información relevante de los demás nodos con el objetivo de coordinar a todos ellos. En esta arquitectura los agentes envían mensajes al nodo central para obtener las direcciones de los nodos que contienen la información requerida, de forma que una vez conocidas las direcciones, no es necesario establecer comunicación con el servidor, solventando una de las limitaciones del modelo maestro-esclavo tradicional. Además, algunas tareas de control pueden distribuirse entre múltiples nodos, teniendo en cuenta que no tienen la capacidad de decisión sobre las acciones a realizar una vez finalicen las tareas asignadas. De igual forma, esta topología sigue presentando un único punto de fallo y limitaciones en cuanto a la escalabilidad del sistema. Un diagrama de topología centralizada se presenta en la figura 2.16a, donde el nodo servidor se presenta en color azul.

2. **Jerárquica:** Esta topología está caracterizada por la presencia de algunos agentes que actúan como super-nodos que tienen control sobre las acciones de otros agentes. En la literatura se encuentran arquitecturas jerárquicas de tres niveles, donde el nivel más alto es el encargado de tomar las decisiones más críticas, los agentes del nivel medio tienen la responsabilidad de tomar decisiones relacionadas con la conexión y desconexión del sistema y el nivel más bajo interactúa con los sensores y los dispositivos. La selección de los super-nodos se realiza de forma dinámica. En la figura 2.16b se muestra un diagrama de estructura jerárquica con los nodos en color azul.

3. **Distribuida:** En una topología de comunicación distribuida (figura 2.16c), cada nodo actúa de forma autónoma, todos tienen el mismo rol en el sistema y se encargan de obtener la información que necesitan de su entorno. Una ventaja significativa que presenta esta infraestructura es precisamente que cada agente puede descubrir información de los otros mediante la comunicación y comunicación con sus vecinos. Esta red no emplea ningún tipo de nodo superior (p.ej servidor central) por lo que evita el problema del único punto de fallo y presenta una alta escalabilidad.

La elección de alguna de las topologías anteriores tiene una implicación directa en la transmisión del flujo de información, en el rendimiento del sistema y la capacidad de expansión de la micro-red. La propuesta centralizada se considera adecuada en la situación en la que los elementos de generación y consumo de la micro-red tienen objetivos similares, por lo que el servidor central puede calcular una estrategia de control global óptima y comunicar las acciones necesarias a los distintos nodos. Sin embargo, una arquitectura descentralizada es más adecuada cuando existen una gran cantidad de elementos de generación y consumo energético para su gestión y monitorización en tiempo real.



(a) (b) (c)

Figura 2.16: Arquitecturas descentralizadas para el control de micro-redes: (a) Centralizada, (b) Jerárquica y (c) Distribuida

2.3.2 Tecnologías de comunicación

El empleo de una determinada tecnología para la transmisión de la información tiene asociados una serie de ventajas y desventajas que se deben tener en cuenta.

Tradicionalmente, las comunicaciones cableadas son las más utilizadas en el intercambio de datos en el contexto de los sistemas eléctricos por su mejor comportamiento, en comparación con las tecnologías inalámbricas, con respecto a su robustez, seguridad y ancho de banda disponible. En contraposición, los despliegues basados en cable suelen incurrir en un mayor coste. Algunas de las tecnologías cableadas más populares utilizadas en el contexto de las micro-redes son las comunicaciones serie *RS-232/422/485* para SCADA, *Ethernet (IEEE 802.3)*, *ModBus*, *ProfiBus* y PLC. En [66] se demuestra que la aplicación de la tecnología *Ethernet* es sumamente adecuado en una micro-red, asegurando un intercambio de información a una tasa lo suficientemente rápida como para el envío/recepción de datos en tiempo real.

Por otro lado, y a pesar de su peor fiabilidad en términos generales, las tecnologías inalámbricas están aumentando sus capacidades de seguridad y pueden ser una alternativa adecuada para la transmisión de información de las distintas unidades pertenecientes a un mismo sistema eléctrico. Algunas de las soluciones más populares son las definidas en el estándar *IEEE 802.15* o en el *IEEE 802.11*, comúnmente conocido como *WiFi*. Este tipo de alternativas tratan de solventar el envío de información en un área de pequeña distancia, aplicado a la comunicación entre los niveles de control primario y secundario de una micro-red.

Por otro lado, las redes celulares (5G, 4G, 3G) se pueden emplear para la implementación de los enlaces de comunicación para el nivel de control terciario o como método de interconexión entre diferentes micro-redes. Una de las principales desventajas de la utilización de las redes móviles como tecnología de comunicación es la disminución en la cobertura por el uso de frecuencias de radio cada vez más altas, lo que hace necesario la instalación de un número mayor de nodos o antenas que pueden incrementar el coste total de la infraestructura. De igual forma, es necesario incorporar una pila de protocolos de seguridad adicional que asegure la confidencialidad de la comunicación, ya que las señales se transmiten de forma libre por el entorno, pudiendo ser captada por cualquier persona o agente externo al sistema.

2.3.3 Comparación sistemas de comunicaciones

Además de la alternativa centralizada SCADA para la implementación de protocolos de comunicación, existe la contraparte descentralizada denominada Multi-Agent System (MAS).

MAS consiste en la integración de múltiples agentes inteligentes que interactúan para resolver problemas de forma conjunta. Estos nodos se estructuran formando alguna de las topologías descritas en 2.3.1 y mostradas en 3.14, donde cada uno de ellos integra una pila de protocolos *TCP/IP*. Las últimas investigaciones indican que las arquitecturas basadas en MAS tienen una serie de limitaciones que dificultan la gestión óptima de las nuevas micro-redes del futuro. La problemática anterior se expone a continuación [65, 67]:

1. Los agentes no pueden comunicarse de forma simultánea con otros agentes, es decir, únicamente se permite la interacción uno a uno entre los nodos. Lo anterior, provoca que ante un fallo de un determinado agente no puede comunicarlo a los demás elementos del sistema hasta que otro agente se comunique con él.

2. La funcionalidad de cada agente individual se puede conocer fácilmente, sin embargo, esa funcionalidad no se puede extender al sistema completo. Para poder conocer el estado global de la micro-red es necesario que un grupo de agentes trabajen de forma conjunta realizando una determinada tarea, sino es muy difícil tener un conocimiento global del estado del sistema.

3. Los nodos no tienen capacidad de reorganizarse de forma dinámica o de auto-reiniciarse, ante fallos locales, *blackouts* de la micro-red o fallos de comunicación.

En la actualidad se siguen investigando nuevos sistemas de comunicación para mejorar las limitaciones anteriores, buscando sistemas dinámicos y complejos que permitan a las micro-redes adaptarse de forma autónoma. La infraestructura Peer-to-Peer (P2P) para el mantenimiento distribuido puede convertirse en el nuevo paradigma de control dinámico de los sistemas eléctricos del futuro.

Una red P2P define una arquitectura de comunicación para sistemas descentralizados, donde los nodos, denominados *peers* pueden actuar como clientes y servidores de forma simultánea. En esta estructura, la conectividad entre los agentes es virtual, es decir, se crean topologías lógicas y estructuradas sobre la red física de comunicación, permitiendo la creación de grupos de *peers* o *clusters* que realizan una determinada tarea de forma conjunta como balanceo de la generación de energía, detección de modo isla y prevención de *blackouts* [68]. No obstante, se debe tener en cuenta que este tipo de arquitecturas aplicadas a las micro-redes continúan en una etapa muy temprana, por lo que es necesario seguir investigando en esta línea.

En la tabla 2.1 se resumen las principales características de la evolución en los sistemas de comunicación aplicados a las micro-redes. Toda la información se ha extraído del estudio realizado en [65].

	Centralizada (SCADA)	Descentralizada (MAS)	Distribuida (P2P)
Acceso a la información	Información del estado de la micro-red a través de todas las unidades	Control independiente con información de los vecinos	Control sobre los vecinos o *clusters*
Estructura de comunicación de datos	Global y síncrona	Local y asíncrona	Local, Global y asíncrona
Coste	Alto	Bajo	Bajo
Modelo de red	Global	Local	Global y Local
Tolerancia a fallos	Muy pobre	1 fallo en una ruta → Aceptable n fallos → Costoso	n fallos → Aceptable
Flexibilidad y modularidad	Se deben reconectar los nodos	Se pueden instalar nodos de forma flexible	Nodos pueden salir y entrar sin cambios en la red
Escala	Pocos nodos	IPv4 → 2^{32} nodos	$>2^{128}$ nodos
Identificación nodos destino	No se permite	Identificación IP única → nodo	Identificador global → nodos
Interoperabilidad	No es posible	Posible	Necesario
Características de red	Alta latencia y ancho de banda limitado No se permite QoS	Mejora el ancho de banda Se permite QoS	Bajas latencias y elevado ancho de banda
Seguridad	Pobre	Solo cuando los nodos son confiables	Posible incluso en situaciones complejas

Tabla 2.1: Comparación de la evolución de los sistemas de comunicación en micro-redes

2.4 Implementaciones prácticas

En el estado del arte actual no se encuentran una gran cantidad de trabajos que lleven a cabo la implementación de algunas de las técnicas que se han descrito en los apartados anteriores, ya que los sistemas que se llevan a cabo en la industria se encuentran englobados en un mercado muy competitivo, sin compartición de conocimiento ni explicaciones en detalle de las arquitecturas *hardware* y *software* desplegadas. En el contexto del control de los convertidores electrónicos de potencia, tanto de forma independiente como integrados en una micro-red, los estudios se realizan en un entorno de simulación. A pesar de ello, en esta sección se describen algunos de los trabajos más relevantes que incluyen una implementación práctica sobre convertidores reales.

El diseño e implementación de los sistemas de control de los convertidores de potencia se han basado, tradicionalmente, en el empleo de FPGAs como el dispositivo de programación *hardware* por excelencia para la creación de periféricos específicos aplicados a las distintas topologías y aplicaciones finales de cada convertidor. De forma general, la FPGA se suele acompañar de un dispositivo de cómputo que añade la suficiente potencia de cálculo para la correcta ejecución del algoritmo de control implementado.

Una de las alternativas como dispositivo de cómputo es la utilización de un Digital Signal Processor (DSP), tal y como se propone en [9], donde se expone la implementación de un algoritmo MPC paralelo para el control de un STATCOM. El artículo detalla un nuevo método de implementación que mejora al método convencional, donde la FPGA es la encargada de controlar el muestreo y obtención de los datos a través de un Analog-Digital-Converter (ADC) y ejecutar los dos algoritmos propuestos de forma secuencial en el DSP.

El nuevo método propuesto consiste en realizar una implementación en paralelo que aproveche al máximo el DSP y la FPGA. El segundo algoritmo (aquel que tiene un mayor coste computacional) se divide en dos partes, de forma que mientras el procesador ejecuta el primer algoritmo, se envían los datos necesarios para iniciar la primera parte del segundo algoritmo sobre la FPGA, consiguiendo una ejecución simultánea. El flujo de implementación del algoritmo se muestra en la figura 2.17.

En la figura 2.18 se muestra el prototipo experimental utilizado con un *STATCOM* basado en un

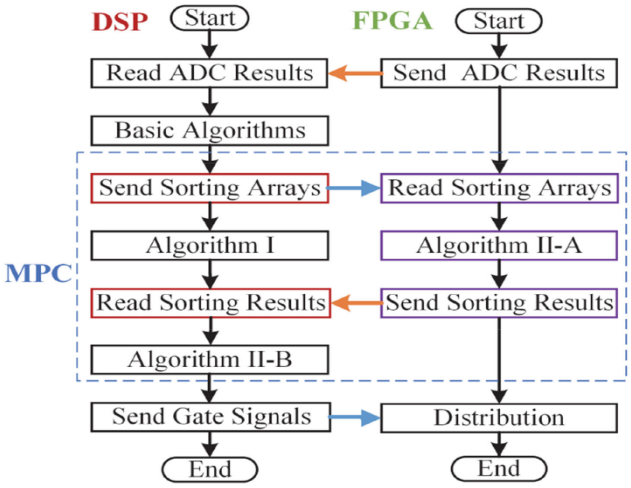


Figura 2.17: Diagrama de flujo del método de implementación utilizado en [9].

convertidor trifásico de 150 V, ± 2.8 kVAr y 7 niveles, donde cada fase consiste en tres celdas, formadas cada una de ellas por un puente en H. La FPGA (*XC6SLX9*) se emplea para muestrear la tensión del bus DC y generar las señales de actuación sobre los IGBTs. La comunicación DSP+FPGA se realiza a través de fibra óptica, donde los algoritmos se implementan en la plataforma de control, basada en una tarjeta que incluye el ADC, la tarjeta del procesador (DSP+FPGA) y dos tarjetas de fibra óptica que permiten comunicar las tarjetas anteriores con cada celda.

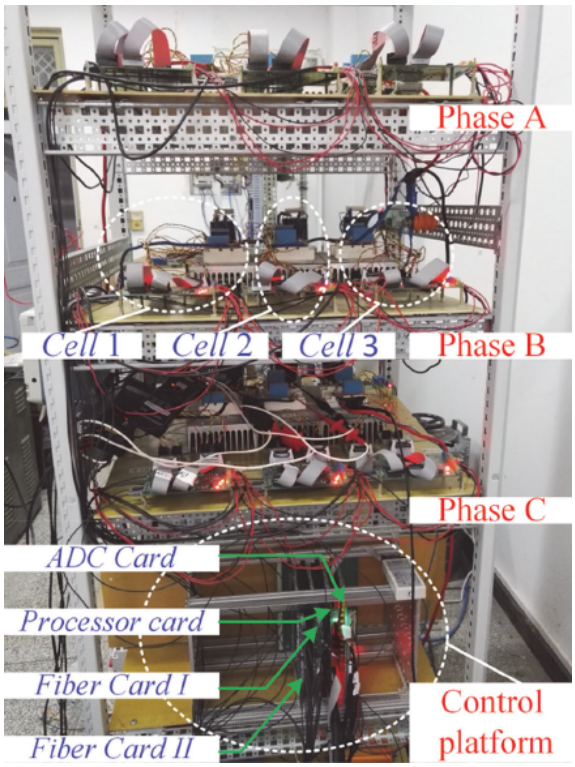


Figura 2.18: Prototipo experimental en [9].

La evolución en las FPGAs y la miniaturización de los dispositivos ha desembocado en el crecimiento del concepto SoC, los cuáles incorporan procesadores de última generación que ofrecen mejores prestaciones. En [10] se describe un esquema de control basado en un observador de perturbaciones *Luenberger*

sobre un motor síncrono de imanes permanentes, implementado sobre la placa *Xilinx Zynq SoC XC7Z020*. La estructura del sistema (figura 2.19) se divide en cuatro partes fundamentales:

1. **FPGA.** Se integra el algoritmo de control de las corrientes y los módulos interfaces como el generador Pulse Width Modulation (PWM), interfaz ADC y el *encoder/decoder*. Los elementos anteriores se diseñan para tener comunicación con el *core* 1 mediante el bus AXI.
2. **CPU1.** Se incluye el módulo de control de velocidad, ya que requiere flexibilidad en su configuración y una buena portabilidad.
3. **CPU0.** Se implementa un sistema operativo embebido como Linux responsable de las comunicaciones con el exterior, es decir, con el ordenador personal del usuario.
4. **On Chip Memory (OCM).** Memoria compartida entre los núcleos CPU0 y CPU1, basada en dos listas circulares.

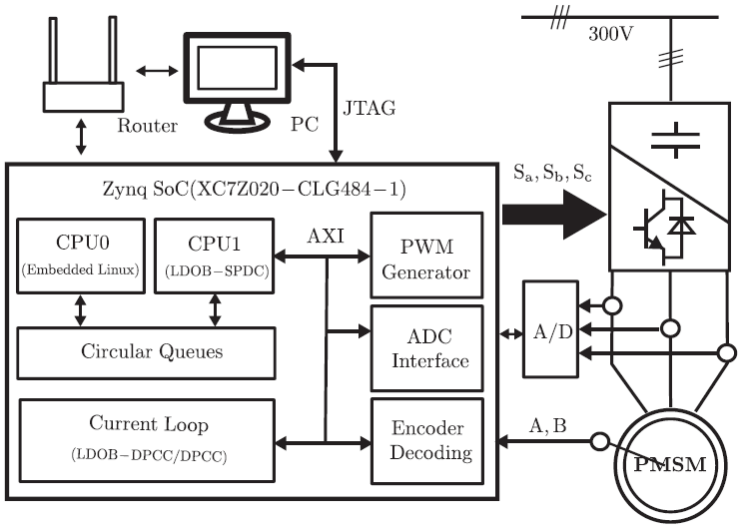


Figura 2.19: Diagrama de bloques del sistema implementado en [10].

De forma similar en [69] se presenta una plataforma para el control de sistemas de electrónica de potencia basada en la implementación en una *Zynq UltraScale+*. Nuevamente, se utilizan los núcleos de la APU para la integración de un sistema operativo y los núcleos R5 pertenecientes a la RPU para la ejecución de algoritmos de control y configuración de los módulos de la FPGA.

Como se deduce de los trabajos descritos, en la literatura no existe una cantidad considerable de implementaciones reales, ya que, o bien los controles no son realizables o no se tienen los medios suficientes para realizarla. Además, se debe tener en cuenta que las propuestas siempre se destinan al control de un único VSC.

2.5 Conclusiones

En este apartado se ha expuesto una descripción de los niveles de control primario y secundario que se encuentran en el estado del arte actual en las micro-redes AC y DC, así como las arquitecturas de comunicaciones que permiten la interacción entre los niveles de control y los distintos elementos que pertenecen a una micro-red.

La contribución de este TFM a la literatura es la preparación de la arquitectura de una plataforma de control y comunicaciones para la optimización del funcionamiento de las siguientes generaciones de micro-redes eléctricas, dotándolas de una mayor inteligencia y contribuyendo a su desarrollo. Para ello, se pretende desarrollar un sistema multi-procesador heterogéneo y asimétrico, basado en la última tecnología de sistemas SoC de *Xilinx (UltraScale+)*, realizando la integración de los algoritmos de control del mayor número posible de convertidores electrónicos de potencia. Se destinan aquellos algoritmos de control más exigentes para los procesadores de tiempo real (RPU) y el resto de algoritmia o servicios en los procesadores APU.

La integración de la virtualización permite disponer de un sistema operativo (*Linux*) encargado de facilitar las comunicaciones con el exterior junto con un hipervisor que gestiona la distribución de máquinas virtuales que ejecutan de forma flexible un sistema operativo o código *baremetal* en función de los requerimientos impuestos por la aplicación concreta. Se trata de una solución mixta entre una arquitectura de comunicaciones centralizada, ya que se dispondrá de un dominio maestro que se comunique con las VM desplegadas para ofrecer al usuario tanto la visualización de información relevante como permitir el control de los convertidores de la micro-red, y una arquitectura distribuida, donde cada VM puede comunicarse con todas las demás para adaptarse a la situación concreta del sistema.

Así mismo, la plataforma puede integrar de forma agregada algoritmos de control secundario, siguiendo alguna de las arquitecturas presentadas en este apartado, en una misma plataforma *multi-core*. Las comunicaciones entre los distintos elementos son internas, a nivel de periodo de muestreo del lazo de control interno, permitiendo tener un mayor control y flexibilidad para actuar ante las perturbaciones severas que pueden sufrir los convertidores de potencia de una micro-red. De esta forma, se puede proporcionar una respuesta coordinada, asegurando una mayor calidad en la energía entregada y garantizando el suministro eléctrico.

Capítulo 3

Estudio teórico

3.1 Introducción

En este capítulo se aborda el estudio de los conceptos teóricos necesarios para la comprensión de este trabajo. En concreto, se presenta una descripción del principio de funcionamiento de los convertidores electrónicos de potencia, la evolución en las topologías, incluyendo las de dos y tres niveles (esta última presente en los convertidores de potencia sobre los que se ha verificado la plataforma de este trabajo) y la explicación de los modos de operación denominados como *Grid Forming* y *Grid Following*. Además, se incluye una introducción a las infraestructuras multi-procesador, centradas en la placa de control *ZCU102* de *Xilinx*, detallando sus componentes principales, tanto aquellos pertenecientes al sistema de procesamiento (PS) como a la lógica programable (PL). Seguidamente, se explican las partes fundamentales de un S.O en una plataforma embebida así como los protocolos de comunicaciones utilizados. Finalmente, se detalla el concepto de virtualización y las diferentes alternativas que presenta para el control de las micro-redes.

3.2 Convertidores electrónicos de potencia

Los convertidores electrónicos de potencia son los elementos básicos para la realización de sistemas de potencia, utilizando dispositivos de conmutación controlados por señales electrónicas para controlar su comportamiento. Existen multitud de criterios para realizar la clasificación de diferentes tipos de convertidores, siendo una de las más comunes la frecuencia de las señales de entrada y salida, aplicado tanto al caso monofásico como al trifásico:

- **Convertidores AC-DC.** También denominados como convertidores rectificadores, a su entrada se dispone de una señal alterna monofásica o trifásica y se consigue como salida una señal continua.
- **Convertidores DC-AC.** Reciben el nombre de convertidores inversores, posibilitan la transformación de una señal continua en una señal de corriente alterna. Son elementos muy flexibles y versátiles porque permiten el control de la frecuencia y el valor eficaz de la señal de salida.
- **Convertidores DC-DC.** Permiten la transformación de los niveles de continua de la señal de salida con respecto a la entrada. Se pueden encontrar ejemplos en los que la tensión de salida es inferior a la entrada (convertidor reductor) o superior a la entrada (convertidor elevador).

- **Convertidores AC-AC.** Este tipo de convertidores se emplean para la obtención de una señal alterna variable a partir de una señal de entrada alterna, pero generalmente fija.

En el contexto de las micro-redes, constituyen los componentes fundamentales actuando de interfaces entre todos los elementos del sistema, posibilitando la integración de un gran número de fuentes de energía con una eficiencia de conversión energética muy elevada.

Este trabajo proporciona una plataforma de comunicaciones y control para cualquier tipo de convertidores monofásicos y trifásicos, sin embargo, la explicación teórica se centra en convertidores inversores/rectificadores trifásicos ya que son sobre los que se ha podido evaluar el sistema.

3.2.1 Principio de funcionamiento

Un convertidor electrónico de potencia AC/DC proporciona una gran flexibilidad al permitir un flujo de potencia reversible, es decir, puede actuar como un rectificador o como un inversor en función de las necesidades que se planteen. Los elementos fundamentales de un convertidor son los dispositivos de conmutación que posibilitan la generación de las señales de salida mediante un controlador que actúa sobre ellos. Un diagrama de bloques general se muestra en la figura 3.1, donde el bloque denominado 'Plataforma de control' es el encargado de la actuación sobre los dispositivos de conmutación del VSC permitiendo, en este ejemplo, la generación de las 3 señales de tensión balanceadas tras un filtro LC.

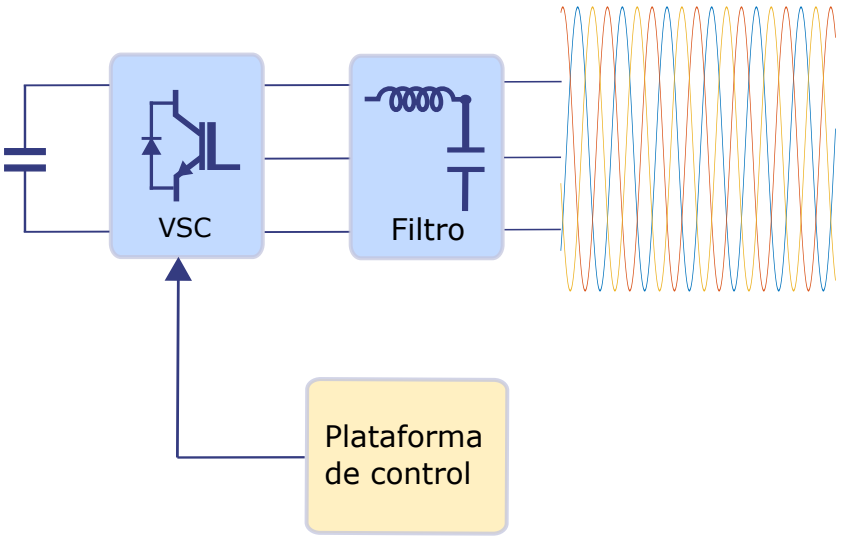


Figura 3.1: Diagrama de bloques del control de un convertidor AC/DC.

El control de los semiconductores de potencia se basa en la aplicación de técnicas de modulación que son totalmente dependientes de la topología del convertidor con el que se trabaja. Aquellas que están basadas en una portadora se denominan Sinusoidal Pulse-Width Modulation (SPWM), consistentes en un proceso de comparación entre una señal moduladora (o señales moduladoras para el caso trifásico) de una frecuencia f_1, generalmente sinusoidal, con una señal portadora triangular de una frecuencia f_2.

Se deben definir los conceptos de modulación en amplitud (m_a) y modulación en frecuencia (m_f). El primero de ellos, especifica la relación en amplitud de pico entre la señal de control o moduladora y la amplitud de la señal triangular (expresión 3.1) y el segundo, representa la relación entre la frecuencia de la señal portadora f_2 y la frecuencia de la señal moduladora f_1 (expresión 3.2):

$$m_a = \frac{\hat{V}_{moduladora}}{\hat{V}_{portadora}} \tag{3.1}$$

$$m_f = \frac{f_2}{f_1} \tag{3.2}$$

Los valores de m_a y m_f son determinantes para la señal de salida, ya que con el índice de modulación en amplitud se puede regular la amplitud del armónico fundamental de salida y con el índice de modulación en frecuencia se ajusta el contenido armónico.

De forma general, el valor de m_f se busca que sea lo suficientemente elevado como para alejar los armónicos, múltiplos de m_f, lo máximo posible de la frecuencia fundamental de la señal de salida. De esta forma, se puede eliminar ese contenido armónico con filtros más pequeños y sencillos de implementar.

Por otro lado, se definen una serie de regiones de funcionamiento en función del valor de m_a, tal y como se muestra en la figura 3.2. En inversores trifásicos se realiza el estudio sobre las tensiones de salida línea a línea (V_{LL}), definiendo tres regiones de operación: lineal, sobremodulación y de onda cuadrada.

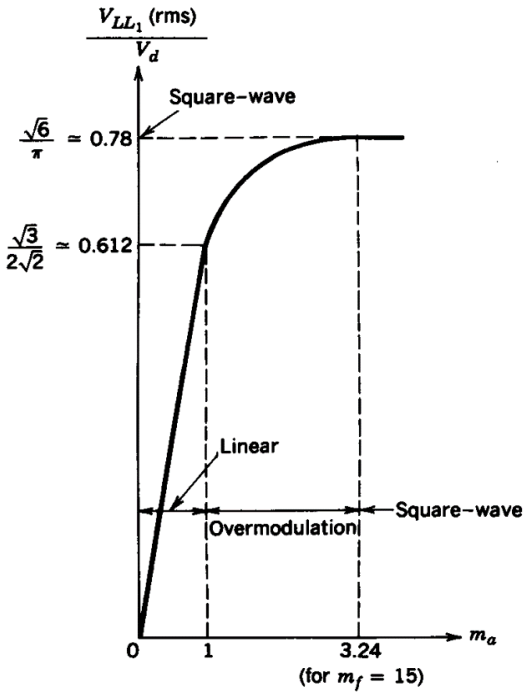


Figura 3.2: Zonas de operación convertidor trifásico. [11]

La zona lineal ($m_a < 1$) es el modo de operación deseado en el contexto de control de convertidores, ya que permite aumentar/disminuir la tensión de forma lineal y controlada. La región de sobremodulación se obtiene con valores más altos de m_a, produciendo tensiones de línea más elevadas pero sin ser proporcionales al índice de modulación en amplitud. Por último, la región de onda cuadrada se consigue con un m_a muy elevado, con una tensión de línea cuadrada a la salida con un valor rms de $0{,}78V_d$, siendo V_d la tensión continua con la que se trabaja.

Se puede observar con claridad como en la zona lineal se realiza una baja utilización del bus DC, llegando hasta un máximo de 61.2 %. Una de las técnicas empleadas para maximizar el rango de la zona lineal es mediante la inserción del tercer armónico o también denominada secuencia cero (expresión 3.3) sobre las tres señales moduladoras, incrementando el aprovechamiento de la tensión continua en un 15.47 %.

$$V_{0s} = [max\{v_{ma}, v_{mb}, v_{mc}\} + min\{v_{ma}, v_{mb}, v_{mc}\}] \tag{3.3}$$

En la actualidad, los convertidores multi-nivel están suponiendo una mejora frente a los convertidores convencionales de dos niveles, ya que permiten generar formas de onda de tensión y corriente más limpias, es decir, con menor contenido armónico. El término 'multi-nivel' define topologías de convertidores con más de una fuente de energía y con la capacidad de obtener más de dos niveles de tensión a la salida. En la figura 3.3 se observa un esquema de los niveles de tensión a la salida en los casos de trabajar con una toplogía de 2 niveles y 3 niveles.

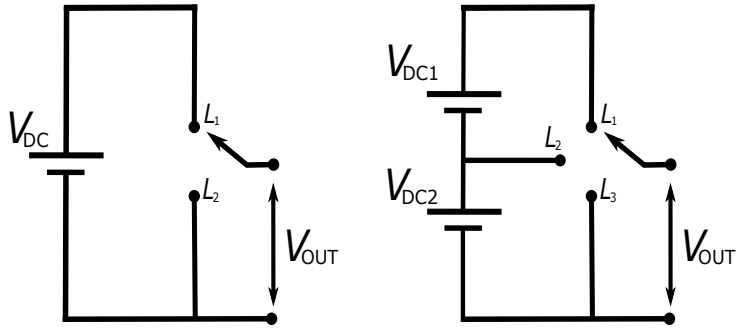


Figura 3.3: Esquema de convertidor de dos niveles (lado izquierdo) y de tres niveles (lado derecho).

Ambas topologías se describen en detalle en los siguientes apartados, empleando como técnica de modulación SPWM.

3.2.1.1 Topología de 2 niveles

En este tipo de topología de convertidores se dispone únicamente de dos niveles de tensión a la salida de cada fase con respecto al punto medio (O) del bus DC. Una representación de un semipuente empleada para una de las ramas de un convertidor trifásico de dos niveles se muestra en la figura 3.4.

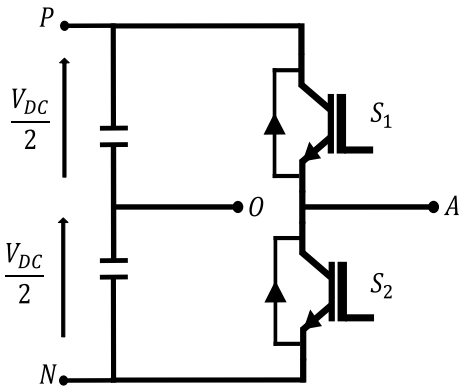


Figura 3.4: Esquema de la rama de la fase A de un convertidor trifásico de dos niveles conectada a un bus DC con punto medio O.

La regla de conmutación de los semiconductores así como el proceso de modulación para una de las fases se incluye en la tabla 3.1 y en la figura 3.5 con un $m_a = 0{,}8$ y un $m_f = 15$ respectivamente, siendo extensible a las otras dos ramas de un convertidor trifásico. Se definen dos posibles estados de los dispositivos de conmutación: el IGBT $S1$ superior se encuentra en ON cuando el valor de la señal moduladora es superior a la señal portadora y por el contrario, se encuentra en OFF cuando la señal moduladora es inferior a la triangular. La regla opuesta se aplica al IGBT $S2$ inferior para evitar cortocircuitar la fuente de tensión o bus DC al que se encuentren conectados.

Comparación de señal	Dispositivos de conmutación	
	S1	S2
$v_{mod} \geq v_{port}$	1	0
$v_{mod} < v_{port}$	0	1

Tabla 3.1: Regla de conmutación para convertidor de dos niveles.

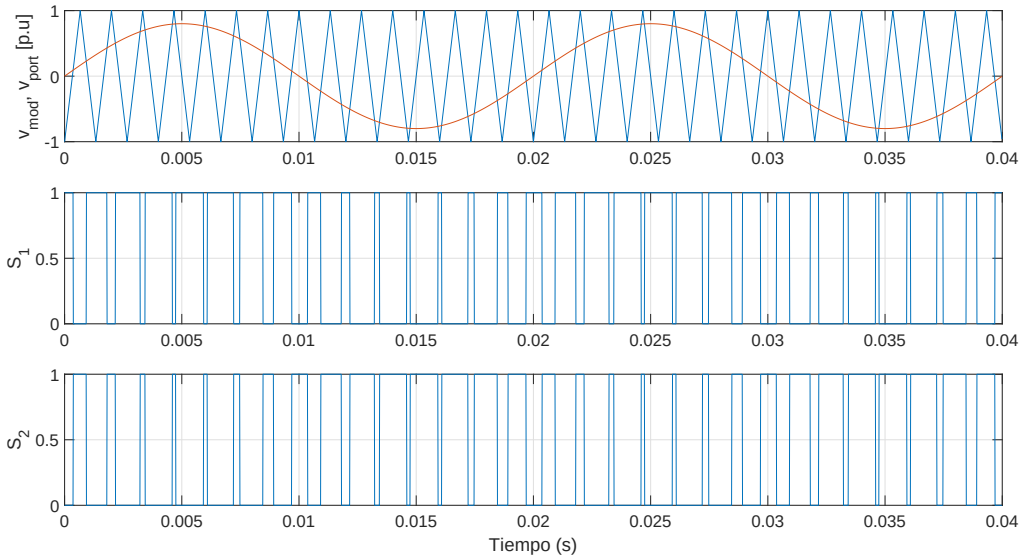


Figura 3.5: Ejemplo SPWM con $m_a = 0{,}8$ y $m_f = 15$

La tensión de salida con respecto al punto medio (v_{AO}) dispone de dos niveles de tensión representados en valores de unidad ($p.u.$) para una mejor comprensión de la forma de la señal de salida (Figura 3.6). El contenido armónico con respecto al armónico fundamental de 50 Hz es bastante elevado como se refleja en el valor de Total-Harmonic-Distortion (THD) de 147.63 %, lo que supone la necesidad de un filtro mucho más selectivo para eliminar los armónicos de orden superior.

Por otro lado, este tipo de construcciones son más sencillas de controlar su comportamiento y suponen un menor coste al ser necesarios muchas menos señales de actuación sobre los dispositivos de conmutación.

3.2.1.2 Topología de 3 niveles

La evolución a convertidores cuya topología permite generar un número más elevado de niveles a la salida es precisamente para disminuir la magnitud de los armónicos de alta frecuencia que distorsionan la señal de 50 Hz, aquella verdaderamente interesante en el contexto de equipos conectados a la red eléctrica.

Esta sección se centra en la topologia 3L-DNPC-VSC, publicada en [70], cuya estructura se adjunta en la figura 3.7. El bus DC se divide en dos partes (representadas como capacidades) con un punto intermedio (O).

De forma similar al proceso de modulación comentado en el apartado 3.2.1.1, se disponen de tres señales moduladoras sinusoidales (una por fase) desplazadas 120º entre ellas, comparando todas ellas con dos señales triangulares portadoras desplazadas en nivel. La regla de conmutación de las señales de activación PWM se detalla en la tabla 3.2, aplicándose a los IGBTs de cada rama con su correspondiente señal moduladora.

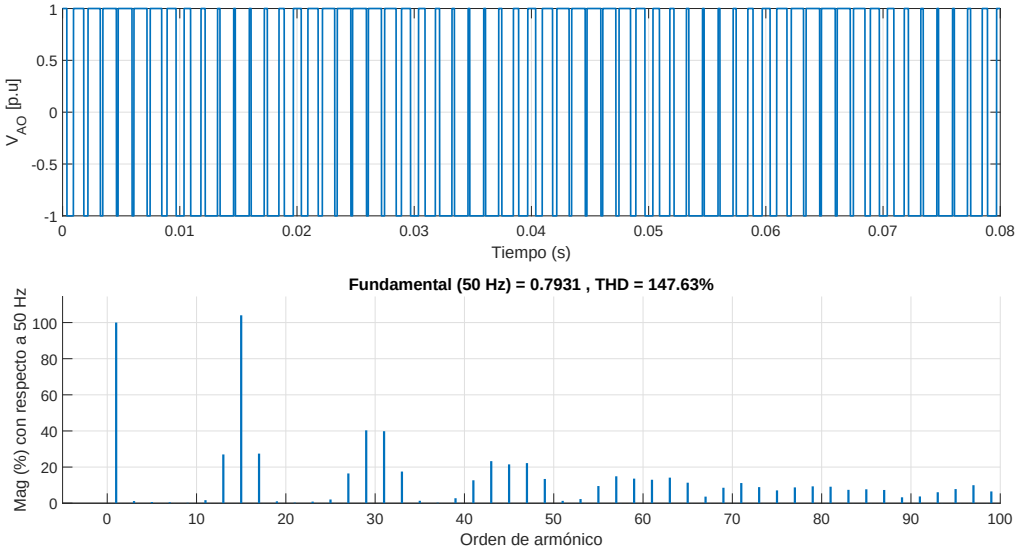


Figura 3.6: Tensión de salida v_{AO} y su contenido armónico para $m_a = 0{,}8$ y $m_f = 15$.

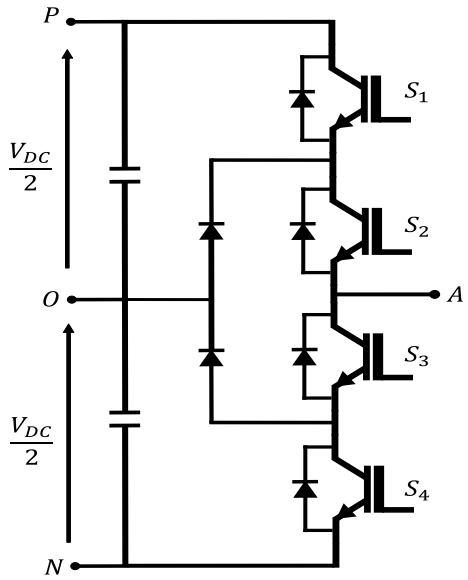


Figura 3.7: Esquema de la rama de la fase A de un convertidor trifásico DNPC.

En la figura 3.8 se muestra el proceso de modulación para la fase A. Las reglas de conmutación se aplican dos a dos, es decir, la señal portadora superior define el cambio de estado de los dispositivos $S1$ y $S3$, mientras que la señal triangular inferior define los estados de $S2$ y $S4$. Se observa con claridad como en cada semiciclo de la señal moduladora, únicamente se encuentran conmutando un par de IGBTs, mientras que el otro par se encuentran con estados complementarios en reposo. Esto permite disponer en la salida (v_{AO}) tres niveles de tensión $-\frac{U_{DC}}{2}$, 0 y $\frac{U_{DC}}{2}$.

En la figura 3.9 se muestra la forma de onda de la señal de salida junto con su contenido armónico. Se observa con claridad una reducción significativa en la magnitud de los armónicos múltiplos de m_f ($THD = 76{,}72\,\%$), permitiendo disponer a la salida de cada fase una señal mucho más limpia, siendo necesario un filtro de menor orden para conseguir la señal fundamental de 50 Hz.

Además de la reducción desde el punto de vista armónico, la estructura DNPC introduce una serie de ventajas con respecto a la de dos niveles [71]:

Comparación de señal	Dispositivos de conmutación			
	S1	S2	S3	S4
$v_{mod} \geq v_{port_1}$	1	-	0	-
$v_{mod} < v_{port_1}$	0	-	1	-
$v_{mod} \geq v_{port_2}$	-	1	-	0
$v_{mod} < v_{port_2}$	-	0	-	1

Tabla 3.2: Regla de conmutación para convertidor de tres niveles DNPC.

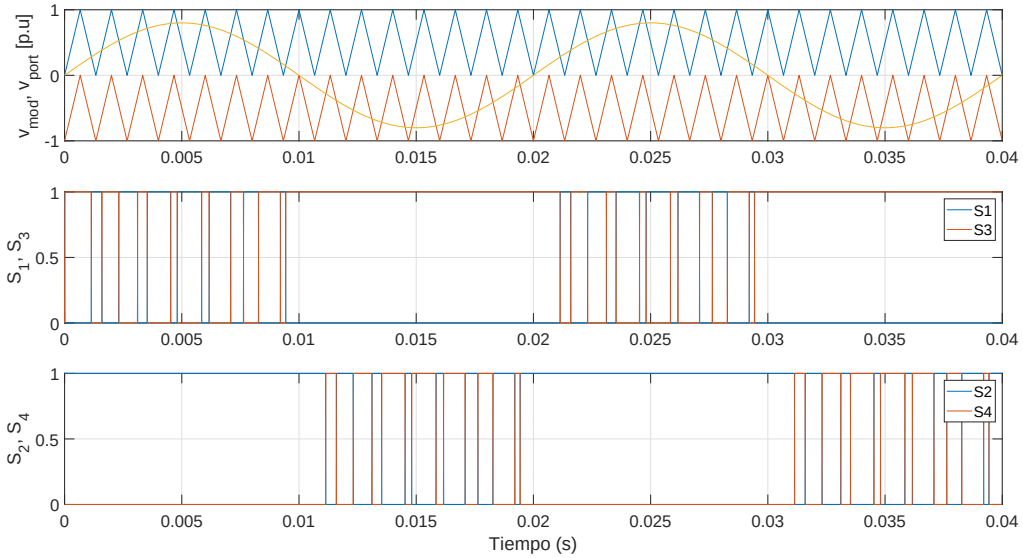


Figura 3.8: Ejemplo SPWM con $m_a = 0{,}8$ y $m_f = 15$.

- **Menores pérdidas de conmutación:** Dado que la conmutación de los dispositivos se realiza con la mitad de tensión, es decir, con respecto a $U_{DC}/2$, supone también la reducción en las pérdidas de cada transistor a la mitad.

- **Menor rizado en la corriente de salida**: La topología NPC permite reducir significativamente el rizado de la corriente de salida, reduciendo la complejidad en el flitrado de salida.

- **Mayor distribución de la tensión del bus DC**: Cada uno de los transistores en abierto mientras se está conmutando soportan la mitad de tensión de la disponible en el bus, permitiendo utilizar dispositivos de conmutación que soporten una menor tensión, disminuyendo el coste total del convertidor.

3.2.2 Modos de operación

Los convertidores electrónicos de potencia pueden operar de diferentes formas, según se encuentren conectados a la red eléctrica o totalmente aislados de ella. El control de estos sistemas consiste en la implementación de lazos de control que permitan proporcionar un funcionamiento flexible en función de las necesidades que se planteen. Se pueden diferenciar dos tipos de modos de operación: *Grid Following* y *Grid Forming*. En la figura 3.10 se muestra un diagrama funcional de los distintos tipos de control que se pueden realizar sobre un convertidor en función de su modo de operación.

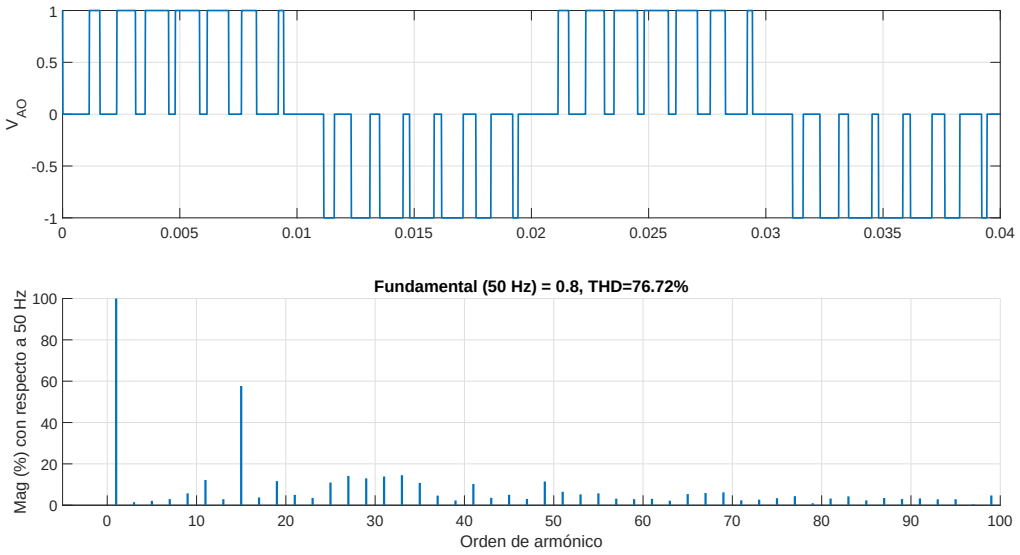


Figura 3.9: Tensión de salida v_{AO} y su contenido armónico para $m_a = 0{,}8$ y $m_f = 15$.

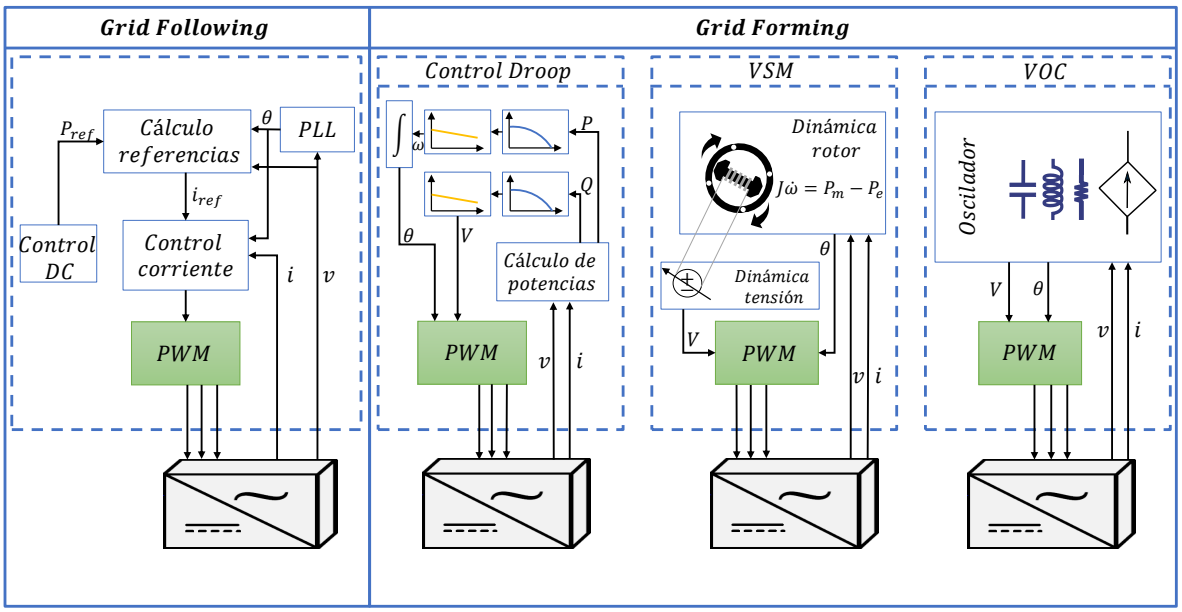


Figura 3.10: Diagrama funcional de convertidores *Grid Following* y *Grid Forming*.

Un convertidor operando como *Grid Following* se modela como una fuente de corriente (figura 3.11a) porque controla la potencia activa y reactiva a través de la amplitud y el ángulo de la corriente inyectada. En este modo de operación no se puede funcionar de forma autónoma sino que se necesita una fuente de tensión que proporcione la amplitud y un ángulo de referencia. Comúnmente esta fuente de tensión suele ser la propia red eléctrica y se realiza la estimación de la fase mediante un Phase-Locked Loop (PLL).

Se debe tener en cuenta que a medida que en una micro-red se integran un mayor número de convertidores actuando como *Grid Following* es posible que sea necesario integrar funcionalidades adicionales que ayuden a mitigar las posibles desviaciones en tensión y frecuencia. Este tipo de funciones se denominan como *Grid Supporting*, las cuáles no son objeto de estudio en este documento.

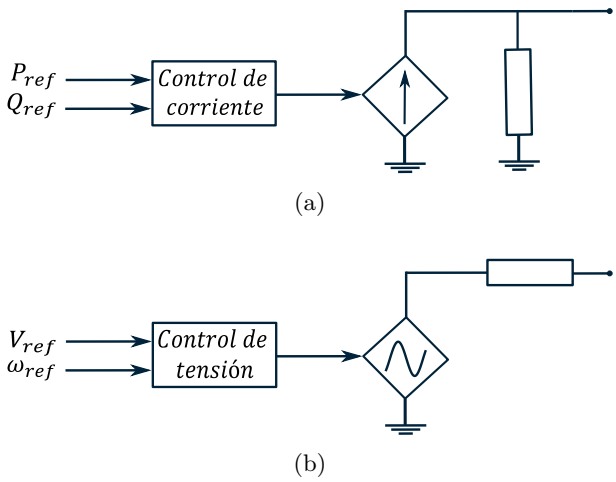


Figura 3.11: Tipos de controles en convertidores: (a) *Grid Following*, (b) *Grid Forming*

Por otro lado, el control *Grid Forming* se modela como una fuente de tensión regulable (figura 3.11b), cuyas referencias son la tensión y la frecuencia a la salida del convertidor. Dado que en este tipo de operación es el convertidor quien 'crea' la red, no es necesario el empleo de un PLL, y es sumamente adecuado para el abastecimiento eléctrico de micro-redes actuando en modo isla, sin conexión a la red eléctrica principal.

Los controles de *Grid Forming* se pueden categorizar como controles *Droop*, Virtual Synchronous Machines (VSM) y Virtual Oscillator Controllers (VOC):

1. **Control Droop**: es el método de control *Grid Forming* que proporciona una relación lineal entre la frecuencia/potencia activa y tensión/potencia reactiva, como una máquina síncrona en régimen permanente. Independientemente de si son máquinas síncronas o convertidores electrónicos cumplen las siguientes propiedades:

   - **Sincronización del sistema:** Todas las unidades alcanzan la misma frecuencia.
   - **Intercambio de potencia:** Cada unidad puede proporcionar un valo de potencia acorde a su capacidad, es decir, a su pendiente de *Droop* programada.

2. **VSM**: Este método fue propuesto en [72] y está basado en la emulación de una máquina síncrona. Las medidas del inversor son las entradas del modelo que emula las dinámicas de una máquina síncrona y proporciona como salidas la referencia de tensión y frecuencia. Una de las posibles implementaciones de este tipo de sistema es el empleo de una inercia virtual que simulan el comportamiento de un rotor y sus relaciones P/ω y V/Q.

3. **VOC**: En los últimos años, otro tipo de métodos de control basados en la emulación de osciladores no lineales están cobrando mucha importancia [73]. De forma similar a la técnica VSM, las medidas se procesan en un modelo que proporciona como salida las referencias de tensión y frecuencia, sin embargo, la principal diferencia se encuentra en la definición del modelo, basado en la emulación de un circuito oscilador cuya frecuencia de resonancia coincide con la nominal de la red eléctrica y sus parámetros se modifican para ajustarse a la tensión de salida deseada.

Finalmente, en la tabla 3.3 se describe desde un punto de vista de alto nivel las principales diferencias entre el modo de operación *Grid Following* y *Grid Forming*.

Grid Following	Grid Forming
Modelo generador de corriente	Modelo generador de tensión
Se asume conexión a la red	Se asume que debe forma la red y mantenerla estable
Control sobre las componentes dq de la corriente inyectada a la red	Control de la tensión y la frecuencia
Control de P y Q desacoplado	Ligero acoplamiento entre P y Q
Necesita PLL	Puede utilizar PLL
Necesita tensión en el PCC para intercambiar P y Q	Puede realizar un *black start* del sistema

Tabla 3.3: Comparación *Grid Following* y *Grid Forming*.

3.3 Multiprocesamiento

Tradicionalmente, las unidades de cómputo se definían como máquinas secuenciales donde el procesador ejecutaba las tareas asignadas instrucción a instrucción, es decir, no se asignaban los recursos de cómputo para realizar la siguiente instrucción hasta que la actual no había finalizado por completo. Las máquinas secuenciales no tienen cabida en la actualidad ya que existen tecnologías y técnicas, tanto a nivel de diseño *hardware* como *software*, para la ejecución de procesos en paralelo.

El concepto de paralelismo define la capacidad de un sistema de ejecutar un conjunto de tareas de forma simultánea, repartiendo todas ellas entre todos los recursos de los que se dispone para incrementar la velocidad de procesamiento. Se pueden encontrar diferentes niveles de paralelismo, desde el paralelismo a nivel de bits e instrucciones hasta los procesos y sistemas de cómputo. Así mismo, el paralelismo se clasifica en función de sobre qué elementos se aplica:

1. **Paralelismo *hardware***: es dependiente del diseño del procesador y está centrado en el proceso de segmentación. La segmentación proporciona la capacidad de descomponer determinadas tareas en operaciones más pequeñas que puedan ser ejecutadas en distintas etapas de procesamiento. Lo anterior, define las arquitecturas en *pipeline* que permiten implementar simultaneidad para mantener todos los recursos disponibles ocupados el máximo tiempo posible. De forma ideal, la velocidad del sistema se incrementa por N, siendo N el número de etapas de segmentación, sin embargo, se deben tener en cuenta la existencia de problemas asociados a las dependencias entre instrucciones y a posibles recursos que se encuentren ocupados por otras etapas. Por último, la mejora en velocidad de la segmentación siempre está limitada por el tiempo de ejecución de la etapa más lenta. En la figura 3.12 se muestra un ejemplo de un sistema aplicando la técnica de segmentación entre múltiples instrucciones.

2. **Paralelismo *software***: se define a nivel de S.O y está basado en la creación de hilos de ejecución. Un proceso puede tener asociado diferentes hilos que comparten entre ellos recursos de computación y de memoria (pudiendo acceder a ella tanto para operaciones de lectura como de escritura). El proceso se encuentra activo mientras alguno de los hilos asociados se encuentre en ejecución. Una vez finaliza el proceso, todos los recursos asociados al mismo son liberados para que se utilice por el propio S.O o para otro proceso en ejecución. En la figura 3.13 se muestra un ejemplo de la ejecución de una tarea dividida en múltiples procesos, los cuáles a su vez se dividen en múltiples hilos.

3. **Paralelismo mixto**: combina el paralelismo *hardware* y *software* mediante la conjunción de múltiples procesadores. El objetivo fundamental es la cooperación de todos ellos para ejecutar múltiples tareas en paralelo, aumentando el rendimiento global de todo el sistema. Se denomina mixto porque en cada procesador se aplican técnicas de segmentación (*hardware*) y se permite la ejecución

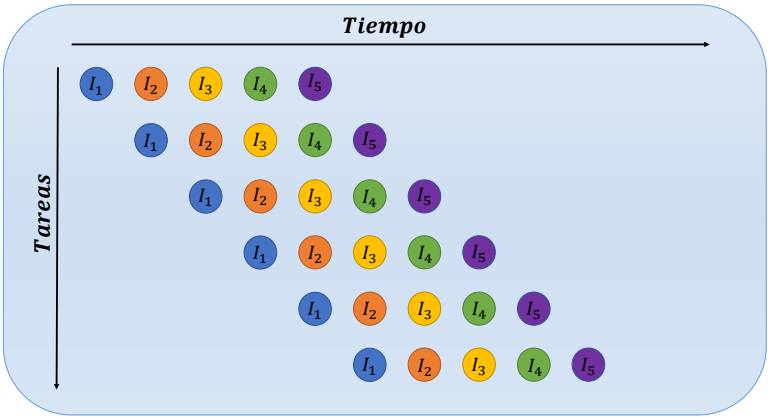


Figura 3.12: Ejemplo de proceso de segmentación.

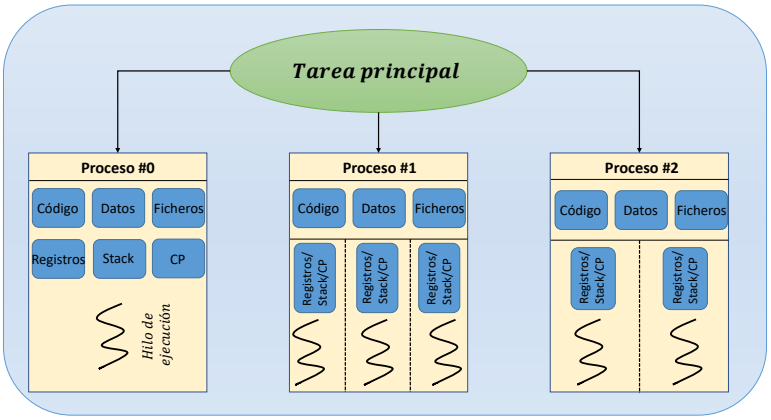


Figura 3.13: Ejemplo de sistema multi-hilo.

de procesos multihilo (*software*). En este contexto, se define el concepto de multiprocesamiento, conjunto de técnicas que permiten la ejecución paralela de una o múltiples tareas. El significado del término difiere en función de la topología de cada máquina concreta:

- **CPU con un único procesador y núcleo.** En este tipo de estructuras el multiprocesamiento se realiza a nivel de proceso, es decir, se dividen las tareas en múltiples procesos que se ejecutan siguiendo la regla marcada por el planificador o *scheduler* del S.O. Los recursos de memoria y periféricos son exclusivos del único núcleo disponible (figura 3.38a).
- **CPU con un único procesador y múltiples núcleos.** El multiprocesamiento se aplica dividiendo la ejecución de las tareas entre los distintos núcleos de la máquina. En este caso, los recursos de memoria y periféricos son compartidos entre los núcleos disponibles (figura 3.38b).
- **CPU multiprocesador y múltiples núcleos.** Similar al caso anterior, se introduce un número mayor de procesadores para aumentar las capacidades de cálculo del sistema. Cada procesador tiene asociado una memoria interna que comparte entre todos sus núcleos (figura 3.14c).

Las relaciones entre los núcleos así como la gestión de los recursos se divide en arquitecturas, las cuáles se describen en detalle en los siguientes apartados.

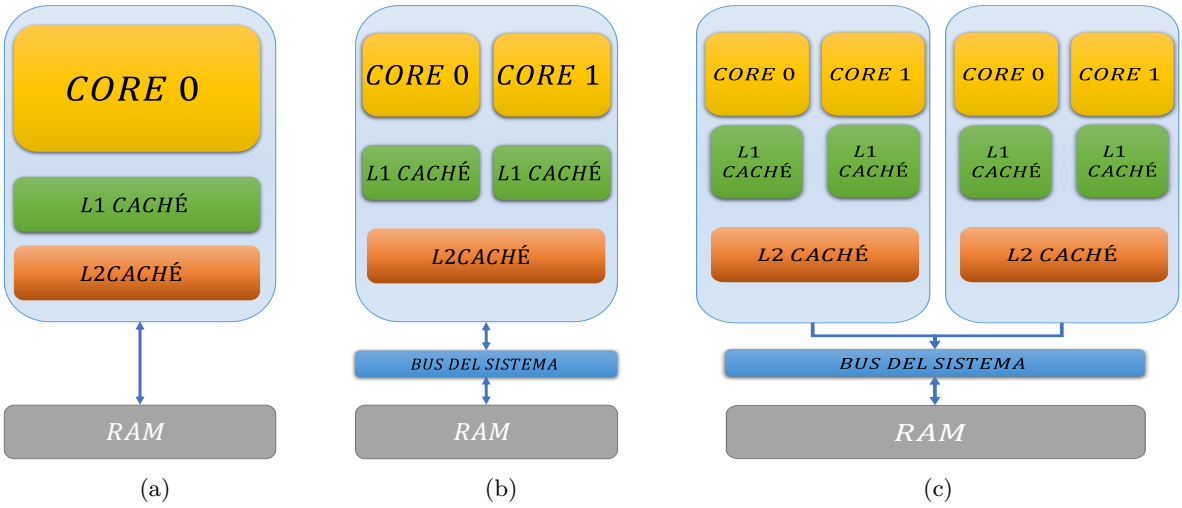


Figura 3.14: Topologías de multiprocesamiento: (a) 1 único procesador y núcleo, (b) 1 procesador con varios núcleos y (c) múltiples procesadores con varios núcleos.

3.3.1 Arquitectura SMP

Una arquitectura Symmetric Multi-Processing (SMP) está basada en el empleo de múltiples núcleos idénticos, los cuáles comparten los recursos (memoria, periféricos...etc) y las tareas a realizar por el sistema (figura 3.15). Se trata de una infraestructura homogénea en la que el coste asociado al acceso a los recursos es el mismo para todos los procesadores o núcleos de la máquina. Por ello, se suele emplear un S.O que arbitra el acceso a la memoria y la ejecución de los distintos procesos. La mayoría de los sistemas comerciales como son los ordenadores de sobremesa, portátiles, *smartphones*...etc, implementan este tipo de arquitectura.

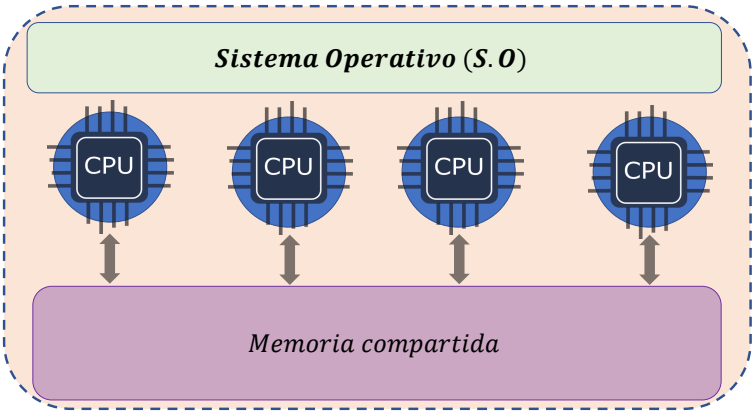


Figura 3.15: Arquitectura SMP.

El inconveniente fundamental de SMP es la compartición de la memoria mediante arbitraje, ya que existen situaciones en las que se quiere acceder a un mismo recurso desde distintos procesadores, la consecuencia es que alguno de ellos se quedará ocioso, es decir, a la espera hasta que el otro núcleo finalice su acceso.

3.3.2 Arquitectura AMP

Una arquitectura Asymmetric Multi-Processing (AMP) se define como una relación maestro-esclavo entre dos o más núcleos conectados físicamente. El núcleo que actúa de maestro es el encargado de controlar el flujo de ejecución de los procesos y las diferentes tareas que desempeñan los otros núcleos del sistema. Esta estructura plantea un paradigma de recursos de computación heterogéneo, donde cada núcleo puede especializarse en realizar una determinada tarea, potenciando la flexibilidad, versatilidad y rendimiento del sistema. La comunicación entre los distintos *cores* se realiza mediante mensajería con un buffer de memoria compartida junto con interrupciones *hardware*.

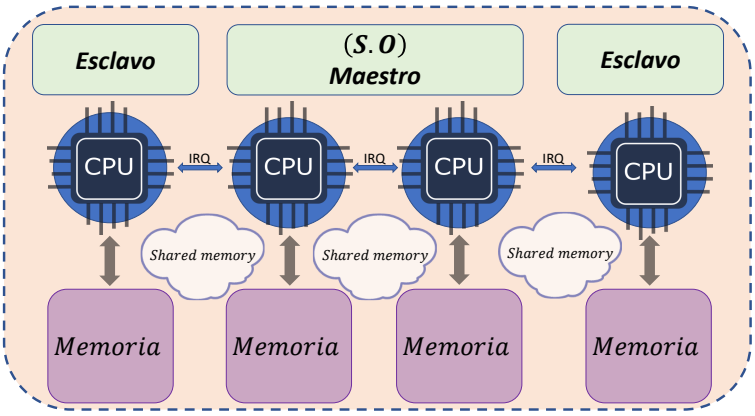


Figura 3.16: Arquitectura AMP.

En contraposición a una arquitectura SMP, no se garantiza que el acceso a los recursos sea equitativo entre todos los núcleos, ya que es el maestro el que decide que recursos asignar a cada uno de los procesadores. En la figura 3.16 se muestra un ejemplo de sistema asimétrico, donde el maestro está formado por un sistema operativo ejecutándose en dos núcleos y los otros dos restantes actúan de esclavos.

Finalmente, en la tabla 3.4 se resumen las principales diferencias entre las dos posibles arquitecturas de compartición de recursos, siendo la alternativa AMP la desarrollada en este TFM para la plataforma de control y comunicaciones para micro-redes.

SMP	**AMP**
Necesidad de S.O para el balance de carga	Puede adoptar múltiples configuraciones, con o sin S.O
Toda la memoria compartida	Cada procesador tiene asignado un área de memoria exclusiva y otra compartida
Puede existir una cola de ejecución de procesos común entre todos los *cores*	El núcleo maestro asigna las tareas a los demás
Sistema homogéneo	Sistema heterogéneo
Comunicación mediante memoria compartida	Interrupciones *hardware*
Buen rendimiento	Rendimiento superior en situaciones específicas

Tabla 3.4: Comparación características *SMP* y *AMP*.

3.3.3 *System on Chip*

Un SoC es un dispositivo que integra un elevado número de componentes de un sistema en un único chip de silicio, consiguiendo así una reducción significativa en el tamaño que ocupa. En la actualidad, ha

cobrado una gran importancia por parte de los fabricantes y desarrolladores de semiconductores como *ARM* y *Xilinx* proporcionando soluciones a los problemas de la industria tecnológica actual.

Este nuevo concepto de sistema integrado permite incorporar componentes digitales, analógicos, una gran cantidad de memoria y periféricos de todo tipo para el desarrollo de aplicaciones con altas exigencias computacionales. A continuación se enumeran algunas de las ventajas que presentan los SoC actuales:

- Mayor rendimiento, velocidad y eficiencia desde un punto de vista computacional gracias a la integración de múltiples núcleos de alto rendimiento.
- Solución de bajo coste comparado a la integración de un mismo dispositivo con las mismas características sobre una placa PCB.
- Redes de comunicaciones ultrarrápidas al disponer de la memoria y de los periféricos en un mismo chip.
- Menor tamaño físico que permite incorporarlo en cualquier espacio de reducidas dimensiones.
- Mayor fiabilidad y menor consumo energético.

Por el contrario, presenta una serie de desventajas que se deben tener en cuenta como son la necesidad de un conocimiento elevado para su manejo, un mayor tiempo en el desarrollo de aplicaciones y una limitada flexibilidad para adaptarlo a ciertas aplicaciones.

En el mundo tecnológico se encuentran ejemplos bastante representativos de SoCs como los que disponen los ordenadores de última generación, móviles, *tablets*...etc, cuyos elementos principales son procesadores de al menos cuatro núcleos, memoria DDR, procesadores gráficos de última generación entre otras funciones.

Precisamente por las limitaciones comentadas con anterioridad, el mercado ha impulsado una solución más flexible denominados SoCs programables. El recurso utilizado es la integración conjunta de una FPGA para dotar al dispositivo de una flexibilidad total, pudiendo reconfigurar el *hardware* en función de la aplicación concreta. En general, se dispone de una parte denominada PS centrada en las unidades de cálculo y de cómputo y la PL o lógica programable para el diseño *hardware*, incluyendo la posibilidad de implementar núcleos utilizando los recursos de la FPGA (*soft cores*).

La siguiente evolución de estos sistemas se denomina MPSoC por la incorporación de un mayor número de *cores* que elevan el rendimiento y las posibilidades de desarrollar aplicaciones más potentes.

Un esquema general de un MPSoC se muestra en la figura 3.17, donde se incluyen múltiples núcleos, GPUs, interfaces de entrada/salida, los relojes, un sistema de gestión de interrupciones (GIC) y la lógica programable.

3.4 Zynq UltraScale+

La implementación del sistema que se describe en este documento se basa en el empleo de un MPSoC+FPGA de la familia *Zynq UltraScale+* desarrollada por *Xilinx*. Este dispositivo combina la potencia de hasta 4 núcleos de alto rendimiento con 2 núcleos de tiempo real basados en la arquitectura ARM, además de un sistema de lógica programable, proporcionando una gran escalabilidad, flexibilidad y capacidad para la ejecución de aplicaciones críticas.

La arquitectura en *UltraScale* proporciona múltiples niveles de rendimiento del sistema en función de las necesidades del desarrollador, con un procesamiento de datos y enrutamiento de señales eficiente

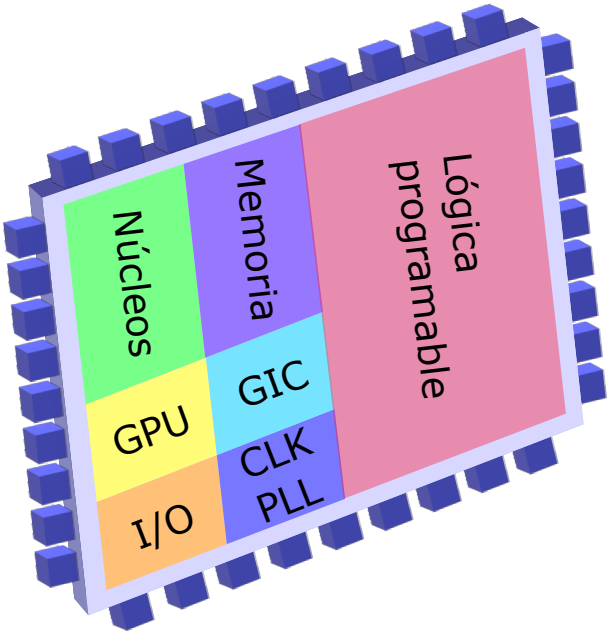


Figura 3.17: Ejemplo de MPSoC + FPGA.

en un mismo chip. Además, incorpora las últimas innovaciones técnicas en la industria como la última generación de *routing* de señales, *ASIC-like clocking*, circuitos integrados *3D on 3D*, la última tecnología en SoC multiprocesador y características de reducción de consumo de potencia [12].

La familia de dispositivos *Zynq UltraScale+ MPSoC* permite el empleo en un amplio abanico de aplicaciones como las que se enumeran a continuación [74]:

1. **Sector automovilístico:** asistencia a la conducción, información al conductor e info-entretenimiento.
2. **Comunicaciones inalámbricas:** soporte para múltiples bandas del espectro radio-eléctrico y desarrollo de antenas inteligentes.
3. **Comunicaciones cableadas:** capacidad de implementar estándares de comunicaciones cableadas y servicios de red.
4. **Centros de datos:** redes definidas por *software* (SDN), pre-procesamiento de datos y análisis.
5. **Visión artificial:** algoritmos de procesamiento de vídeo, detección de objetos...etc.
6. **Electrónica de Potencia:** algoritmos de control y comunicaciones para el desarrollo de la lógica de control en entornos industriales.

En la misma familia se disponen de distintos dispositivos que contienen el mismo PS y PL, variando únicamente si se dispone de bloques *hardware* de procesamiento de vídeo así como disponibilidad de GPUs. En la figura 3.18 se adjuntan los componentes de cada una de los dispositivos de la familia que proporciona Xilinx (en color rojo se señala el dispositivo integrado en la placa de evaluación *ZCU102* utilizada para el desarrollo del proyecto).

Un diagrama general de la infraestructura presente en la arquitectura *Zynq UltraScale+* se muestra en la figura 3.19, donde se observa la conjunción entre la PS, formada por la APU, la RPU y la GPU así como la memoria y los periféricos que permiten la comunicación con el exterior, y la PL, con bloques lógicos que permiten la implementación de cualquier diseño *hardware*.

	CG Devices	EG Devices	EV Devices
APU	Dual-core Arm Cortex-A53	Quad-core Arm Cortex-A53	Quad-core Arm Cortex-A53
RPU	Dual-core Arm Cortex-R5F	Dual-core Arm Cortex-R5F	Dual-core Arm Cortex-R5F
GPU	–	Mali-400MP2	Mali-400MP2
VCU	–	–	H.264/H.265

Figura 3.18: Comparación dispositivos familia *Zynq UltraScale+*.

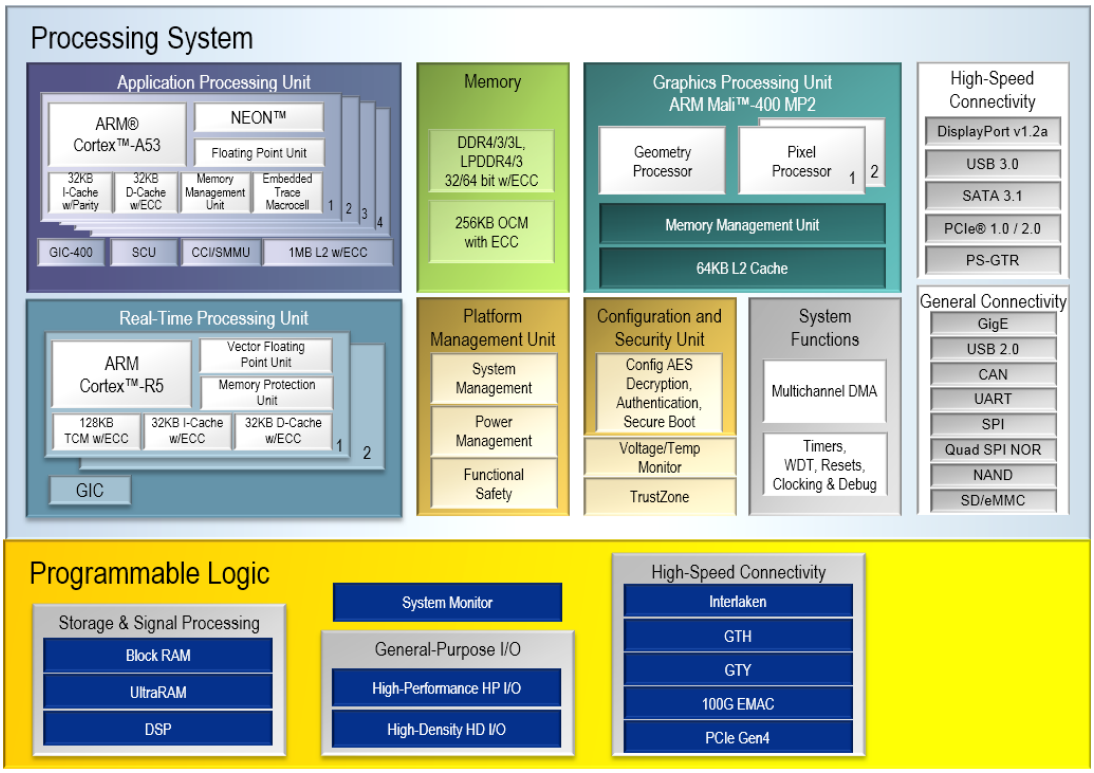


Figura 3.19: Arquitectura *Zynq UltraScale+*. Fuente: www.xilinx.com

Una descripción detallada de los distintos elementos presentes en el chip se exponen en los siguientes apartados.

3.4.1 Processing System (PS)

El sistema de procesamiento o PS presente en el chip *Zynq UltraScale+* se puede dividir en tres grandes subsistemas [12]: la APU, la RPU y la Platform Management Unit (PMU).

3.4.1.1 APU

El subsistema APU (figura 3.20) está formado por cuatro procesadores *Cortex-A53* con dos memorias caché de nivel L1 (para datos e instrucciones) exclusivas para cada núcleo y una memoria caché L2 compartida entre todos ellos. Los procesadores A-53 están basados en la arquitectura ARM-v8, proporcionando una eficiencia energética superior y con la capacidad de ejecutar código de 32 y 64 bits. Además, consta de un sistema multi-etapa para implementar técnicas de paralelismo avanzado para la búsqueda y el acceso a los datos.

Cada uno de los *cores* dispone de las características que se listan a continuación:

- Estados de ejecución compatibles con arquitecturas de 32 y 64 bits.
- Niveles de excepción (EL0, EL1, EL2 y EL3) para cada estado de ejecución.
- Empleo del conjunto de instrucciones basadas en la arquitectura ARM-v8 incluyendo extensiones para operaciones en coma flotante (VFPv4) y para aplicación de técnicas de criptografía.
- Cachés de nivel 1 (L1) independientes de 32 KB para instrucciones y datos.
- Unidad de gestión de memoria (MMU) de dos etapas (una etapa se asigna a un posible hipervisor y la otra para los *guests*).
- Memoria caché de nivel 2 (L2) de 1 MB compartida por todos los núcleos. Incorpora una unidad que mantiene la coherencia de los datos.
- Accelerator Coherency Port (ACP).
- Arquitectura ARM-v8 de depuración.
- Soporte para virtualización *hardware* que permite el despliegue de múltiples dominios *software* sobre la misma plataforma.
- Unidad de criptografía *hardware* que incrementa el rendimiento x10 comparado con las operaciones criptográficas en *software*.
- Alta capacidad de direccionamiento con el empleo de una memoria de 4GB.
- Tecnología *TrustZone* para la implementación de aplicaciones en modo seguro.
- Unidades de procesamiento NEON para soporte SIMD y unidades de computo de coma flotante FPU.

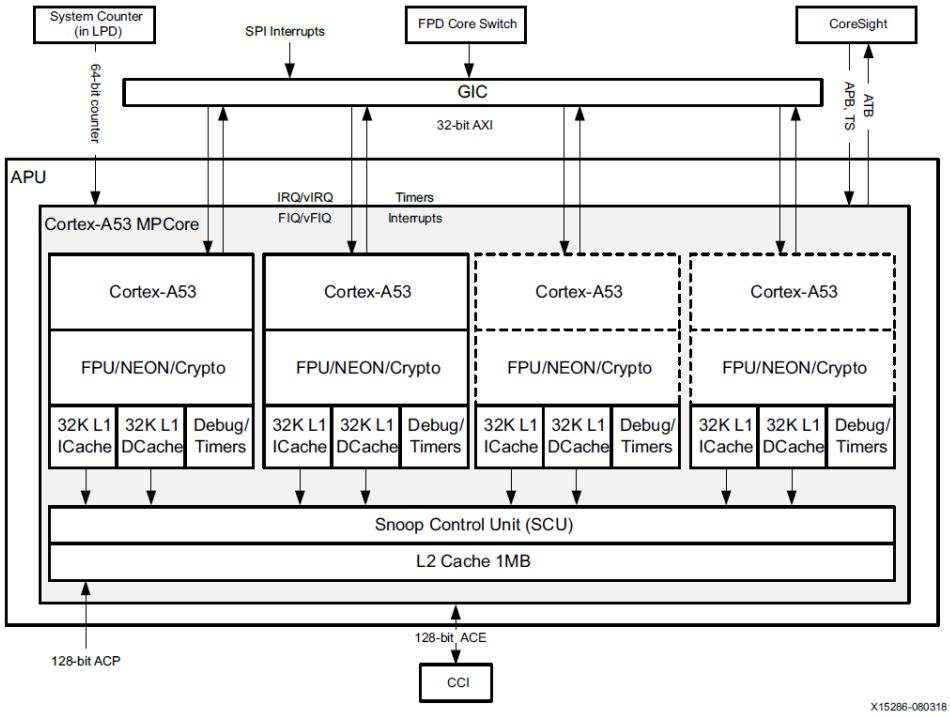


Figura 3.20: Diagrama de bloques APU [12].

El modelo de excepción en la arquitectura ARM-v8 define tres niveles de excepción (un nivel más ato indica mayores privilegios para la ejecución de código):

1. **EL0** tiene el nivel más bajo de privilegio de ejecución *software*, comúnmente se le suele denominar con nivel de ejecución no privilegiado.
2. **EL1** proporciona el soporte para el estado básico de ejecución no seguro.
3. **EL2** permite añadir la funcionalidad de virtualizar los procesadores
4. **EL3** está destinado al soporte de las aplicaciones en modo seguro.

Además de todo lo anterior, cada uno de los núcleos dispone de contadores exclusivos para su propia temporización, así como un sistema de interrupciones general que se encarga de distribuir las fuentes de interrupción entre los distintos procesadores de la APU. La comunicación con los recursos de la PL se realiza a través de un bus multi-capa denominado *AMBA AXI Interconnect* mediante el protocolo AXI.

En el sistema desarrollado en este trabajo, los *cores A-53* se utilizan para albergar un sistema operativo como *Linux* que proporcione los servicios de comunicación con el exterior y para la integración de un hipervisor que habilite la virtualización y generación de máquinas virtuales.

3.4.1.2 RPU

El subsistema RPU (figura 3.21) está formado por dos procesadores *Cortex-R5F* para el procesamiento en tiempo real. Estos núcleos implementan una arquitectura ARM-v7 de 32 bits junto con una unidad de coma flotante (VFPv3).

La latencia de las interrupciones se mantiene en niveles muy bajos gracias a la forma de atender a las mismas (se dispone de un puerto dedicado de baja latencia con conexión directa con el controlador de interrupciones) y accesos a la memoria determinísticos y ultrarrápidos mediante puertos de baja latencia dedicados a la RAM local de cada núcleo. Las características de los procesadores *Cortex-R5F* se listan seguidamente:

- Conjunto de instrucciones de 32 bits pertenecientes a la arquitectura ARM v7-R.
- Subsistema de cálculo de doble precisión FPU con instrucciones VFPv3.
- Interfaz maestra de 64 bits AXI3 para el acceso a memoria y periféricos.
- Interfaz esclava de 64 bits AXI3 para el acceso a DMA a las memorias exclusivas TCM.
- Bancos de memoria TCM separados de 128KB con protección frente a errores ECC.
- Configuración de ejecución de procesadores de forma independiente o de forma redundante (*dual redundant*).
- Cachés L1 de 32 KB para instrucciones y datos con protección ECC.
- Unidad de monitorización de rendimiento.
- Watchdog para detectar tanto errores sistemáticos como fallos aleatorios en el flujo de ejecución de un programa.

Este subsistema se reserva para la ejecución de los algoritmos de control de los convertidores pertenecientes a una micro-red, ya que requieren el cumplimiento de ciertos requisitos de tiempo real que permitan un correcto funcionamiento del sistema.

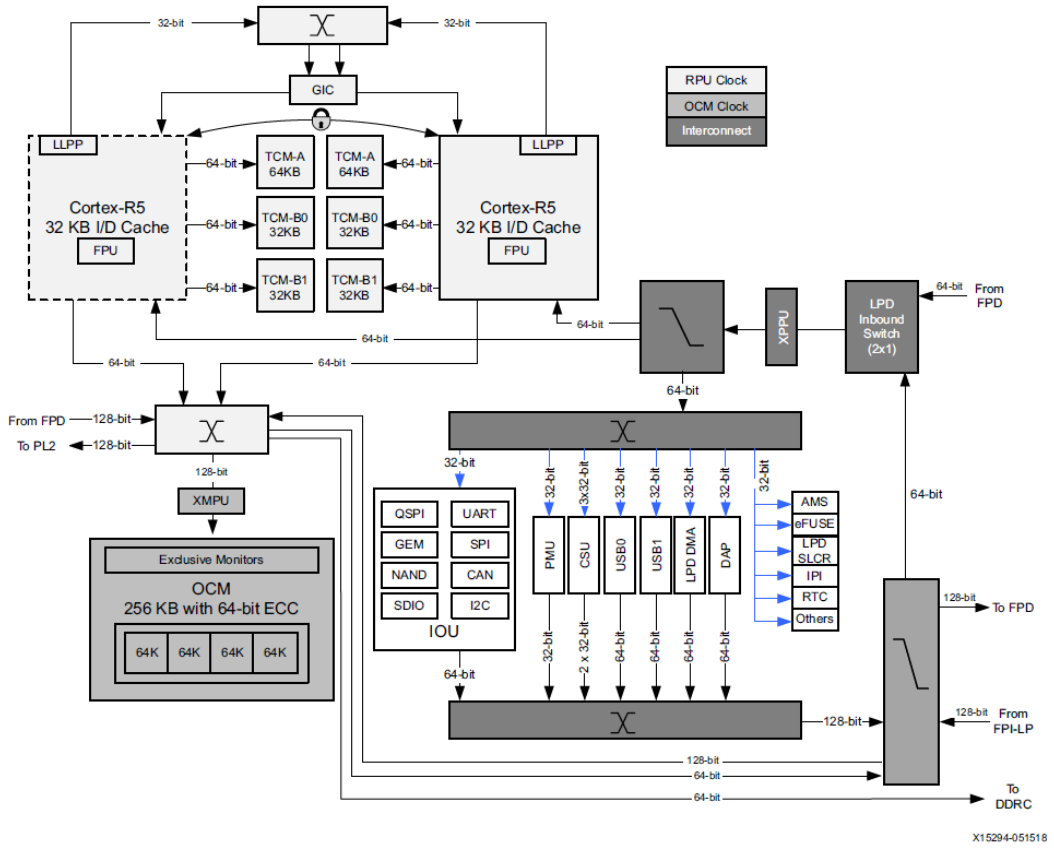


Figura 3.21: Diagrama de bloques RPU [12].

3.4.1.3 PMU

Zynq UltraScale+ incorpora un procesador programable por el usuario (PMU) para la gestión de la potencia, los errores y la ejecución de librerías de *test software* para las aplicaciones. Las tareas fundamentales que realiza la PMU son las siguientes:

- Inicialización del sistema antes del arranque.
- Gestión de la potencia consumida.
- Ejecución de librerías de test (opcional).
- Sistema de atención a los errores.

Se definen cuatro dominios de potencia (figura 3.22), tres asociados al PS y otro a la PL. A continuación se describen los relativos al PS:

1. **Dominio de batería**: este modo permite mantener la información crítica almacenada a lo largo del tiempo cuando el dispositivo se encuentra apagado. Los bloques BBRAM y el reloj de tiempo real (RTC) son los que pertenecen a este dominio.
2. **Dominio *low power***: este modo está formado por la unidad RPU, la memoria *on-chip* (OCM), la PMU y periféricos de baja velocidad.
3. **Dominio *full power***: este dominio está formado por la APU, la Graphics Processing Unit (GPU), el controlador de memoria DDR y periféricos de alto rendimiento como *PCI Express, USB 3.0, Display Port y SATA*.

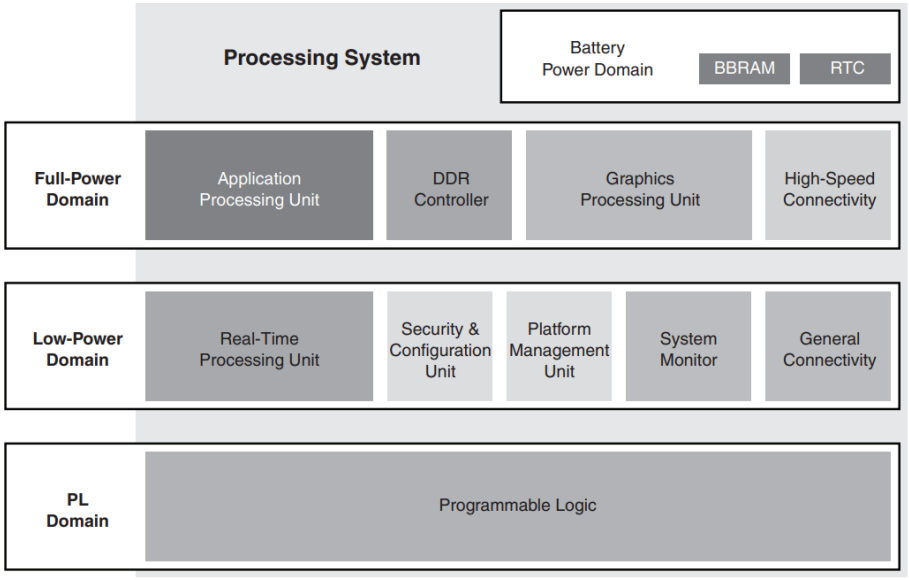


Figura 3.22: Dominios de potencia MPSoC *Zynq UltraScale+* [13].

Una de las principales ventajas que ofrece este subsistema es la posibilidad de decidir los dominios de potencia activos, mejorando el ratio $rendimiento/watios_consumidos$ y hacer uso de los recursos disponibles de una forma más eficiente. En la figura 3.23 se representa la relación entre el nivel de actividad del chip y el consumo de potencia requerido.

Un ejemplo de optimización en la gestión de potencia es la asignación de las tareas al procesador correcto. En general, las tareas relacionadas con el procesamiento de datos se suelen asignar a los núcleos *Cortex A-53*, sin embargo, si algún procesador *Cortex-R5* se encuentra inactivo, se debe sopesar la integración de esas tareas en ese *core* porque el consumo energético es bastante inferior al pertenecer al dominio *low power*.

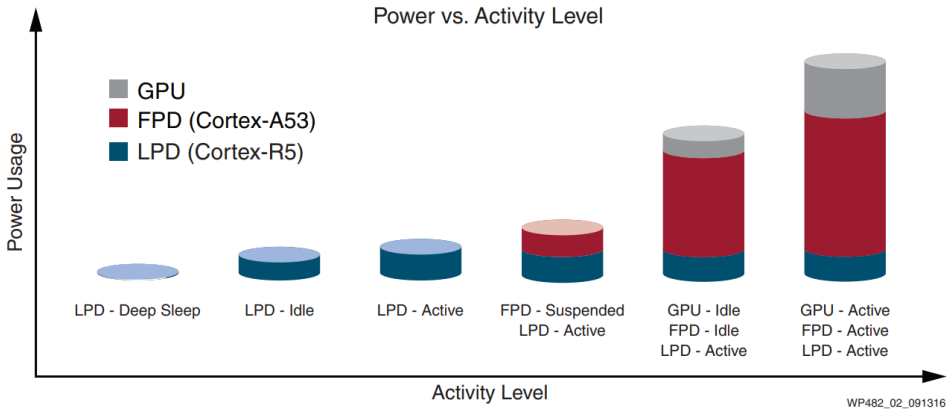


Figura 3.23: Ejemplo de modos de potencia [13].

3.4.2 Programmable Logic (PL)

La segunda parte fundamental del MPSoC basado en *Zynq UltraScale+* es la lógica programable o PL (figura 3.24). La FPGA integrada está formada mayoritariamente por lógica de propósito general, compuesta a su vez por bloques lógicos configurables (CLBs) y bloques de entrada y salida (IOBs).

Los elementos principales de la PL se resumen a continuación [14]:

- ***Configurable Logic Block (CLB):*** se definen como pequeñas agrupaciones regulares de elementos lógicos distribuidos en forma de una matriz de dos dimensiones, conectados todos ellos entre sí a través de conexiones programables.
- ***Slice:*** sub-elemento de un CLB que contiene recursos para la implementación de circuitos combinacionales y secuenciales.
- ***Lookup Table (LUT):*** componente capaz de implementar (i) funciones lógicas de hasta 6 entradas, (ii) una memoria ROM pequeña, (iii) una memoria RAM pequeña o (iv) un registro de desplazamiento. Por supuesto pueden combinarse varias LUTs para generar funciones lógicas, memorias y registros más grandes y complejos.
- ***Flip-flop (FF):*** circuito secuencial que implementa un registro de 1 bit con la funcionalidad de un *reset*.
- ***Matriz de conmutación:*** matriz de conexión ubicada en cada lado de un CLB, proporcionando una solución de interconexión flexible entre elementos.
- ***Bloques de entrada/salida (IOBs):*** interfaces entre los elementos de la lógica y los pads físicos. Se ubican en la periferia de la FPGA.

Además, la PL incorpora recursos de propósito especial como bloques de memoria RAM, *slices* aritméticos de alto rendimiento (DSP48E1) así como conectividad con el exterior mediante un puerto JTAG para facilitar la configuración y depuración de la lógica programada.

La finalidad de la FPGA en la plataforma de control y comunicaciones de este proyecto es la implementación de IPs (*Intelectual Properties*) que generen las señales de control sobre los convertidores electrónicos de potencia.

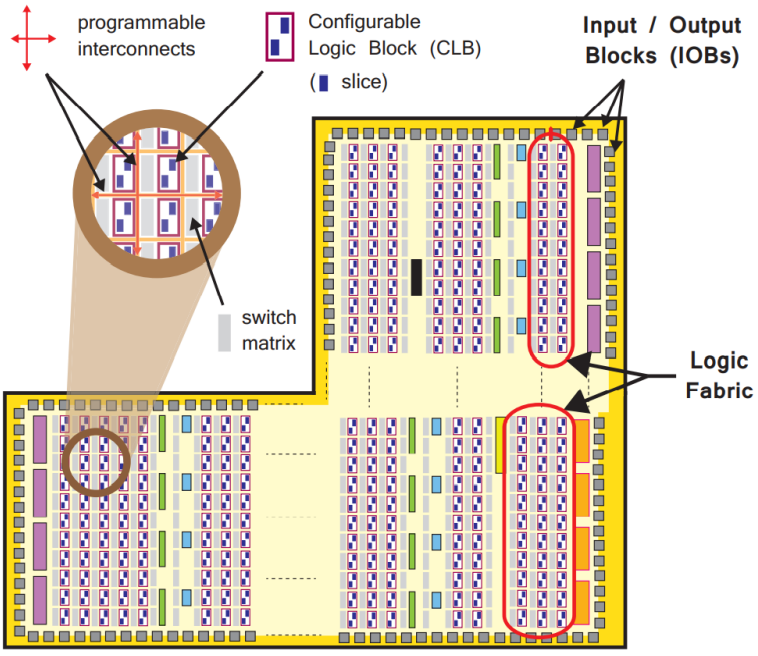


Figura 3.24: PL y sus componentes del chip *Zynq UltraScale+* [14].

3.5 Linux embebido

La integración de un S.O sobre una plataforma embebida proporciona una gran flexibilidad para el desarrollo de aplicaciones complejas que requieran comunicación e interacción por parte del usuario, así como proveer un mecanismo de multitarea sobre el *hardware* subyacente.

Uno de los *software* de código abierto más conocidos para su implementación en este tipo de plataformas es Linux gracias a las múltiples ventajas que presenta:

- Se reduce el coste del despliegue *software* (gratuito).
- Control total sobre el código fuente y los elementos a integrar.
- Un gran desarrollo de componentes que permite mejorar la calidad global del sistema operativo.
- Facilidad para evaluar las aplicaciones.

Todo sistema embebido *Linux* está formado por una serie de componentes fundamentales, cuya funcionalidad se describe en detalle a continuación:

1. ***Bootloader:*** una vez el sistema tiene alimentación y está encendido, se carga en memoria un elemento denominado *bootloader*, cuya finalidad es buscar el programa binario del sistema operativo, alojarlo en memoria y comenzar su ejecución. El programa binario anterior no es más que el núcleo de Linux. El *bootloader* no volverá a ejecutarse hasta que el sistema se reinicie.
2. ***Kernel:*** el núcleo o *kernel* se encarga de ejecutar códigos de inicialización entre los que se incluye la configuración del *hardware*, creación de estructuras de datos del sistema, iniciar el planificador (*scheduler*), los *drivers*, montar el sistema de ficheros y ejecutar el primer programa de inicio, el cual se encarga de lanzar los servicios por defecto del sistema y los procesos que hayan sido configurados para ejecutarse al comienzo.
3. ***Sistema de Ficheros:*** en *Linux*, así como en cualquier S.O moderno, los datos y los programas a ejecutar se encuentran distribuidos de forma jerárquica en ficheros. Este sistema suele estar almacenado en el disco duro principal del ordenador, sin embargo, las plataformas embebidas al tener unas dimensiones reducidas, suelen emplear dispositivos de almacenamiento como SD, memorías *flash* o la propia memoria RAM.
4. ***Servicios y aplicaciones:*** combina todos aquellos programas que permiten dotar de utilidad al sistema, en general, los servicios suelen ir asociados a tareas en segundo plano como la gestión de redes, el montaje de sistemas de ficheros adicionales, el entorno gráfico..etc, mientras que las aplicaciones son aquellas desarrolladas por/para el usuario con multitud y diferenciadas funcionalidades en función del contexto concreto donde se apliquen.

La creación de aplicaciones para una determinada plataforma es dependiente de la arquitectura concreta del dispositivo, siendo necesario un elemento denominado *Toolchain*, formado por un conjunto de herramientas que permite generar programas compatibles con la arquitectura de la máquina destino. En la figura 3.25 se muestra un esquema del flujo de trabajo que se sigue para la integración de un S.O sobre un SoC.

En el desarrollo del sistema final implementado en este TFM se utiliza como conjunto de herramientas agrupadas en un programa denominada *Petalinux* de *Xilinx*.

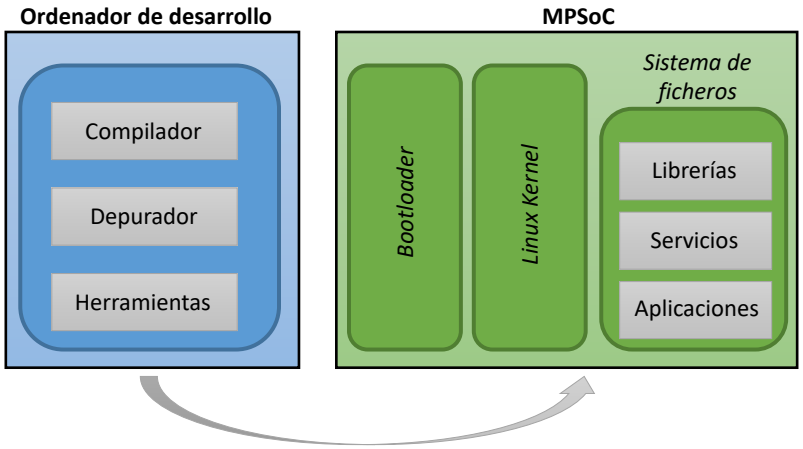


Figura 3.25: Flujo de trabajo para la integración de un sistema operativo en un MPSoC.

3.5.1 Petalinux

Petalinux define un conjunto de herramientas de desarrollo para la integración de una distribución *Linux* sobre los sistemas SoC ofreciendo la posibilidad de personalizar, generar y desplegar soluciones embebidas para acelerar los tiempos de diseño sobre las familias *Zynq UltraScale+*, *Zynq* y *MicroBlaze*. Las herramientas principales que incluye *Petalinux* son las siguientes:

(i) Una interfaz de línea de comandos.

(ii) Constructor del sistema de arranque de *Linux*.

(iii) Herramientas de compilación cruzada y automatización.

(iv) Creador automático de aplicaciones, *drivers* y librerías.

(v) Simulador integrado de un sistema completo (QEMU).

El flujo de diseño de *Petalinux* siempre parte de un fichero de descripción de la plataforma *hardware* (*Hardware Description File*), obtenido a través del programa de diseño *Vivado* de *Xilinx*. Se pueden diferenciar varias etapas en el proceso completo de integración del núcleo de *Linux* sobre la plataforma embebida, donde cada una de ellas tiene una correspondencia directa con un comando de *Petalinux* (figura 3.26).

Una vez exportado el fichero de descripción *hardware* se selecciona la plataforma sobre la que se pretende integrar el S.O (en este trabajo se ha seleccionado *Zynq UltraScale+*). Seguidamente, se personaliza el sistema con los componentes que se requieran tanto en el *kernel* como en el sistema de ficheros (aplicaciones, librerías, servicios...etc). Posteriormente, se realiza el proceso de compilación de todo el sistema para generar la imagen que agrupe el *kernel* y el sistema de ficheros anteriormente configurado. A partir de este momento, el desarrollador tiene dos opciones: la ejecución del sistema sobre la plataforma de simulación incluida (QEMU) o empaquetar todo en ficheros de arranque para incluirlo en algún dispositivo de almacenamiento y verificar el funcionamiento sobre el *hardware* real.

3.5.2 *Device Trees*

Un *Device Tree* es una estructura de datos de descripción *hardware* que permite al núcleo de un S.O utilizar y gestionar los distintos componentes como la CPU, la memoria, los buses y los periféricos del

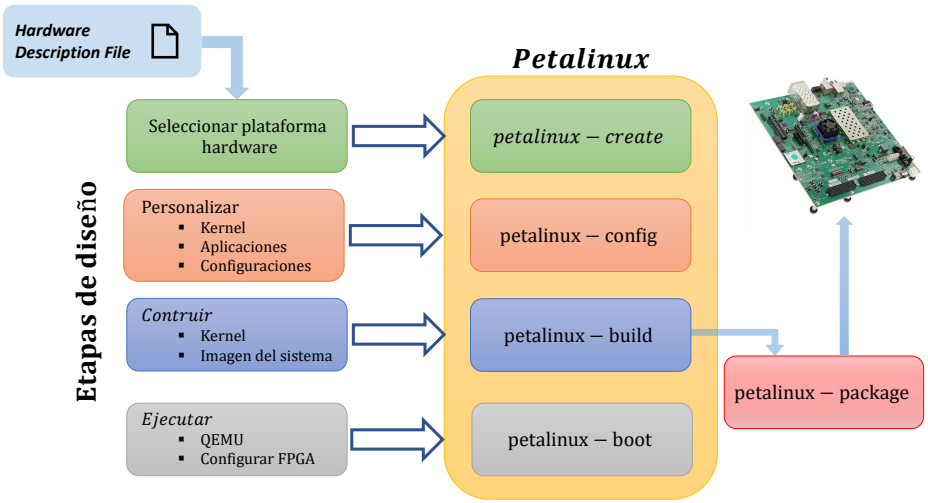


Figura 3.26: Etapas de diseño en *Petalinux*.

sistema.

El *kernel* de Linux necesita saber los detalles de la topología *hardware* de la máquina antes de poder cargar los *device drivers* que controlen el acceso a los distintos dispositivos.

En este contexto, es labor del desarrollador describir los dispositivos *hardware* de los que dispone el sistema en un fichero de configuración denominado Device Tree Source (DTS) con un aspecto similar a un archivo de tipo XML como se muestra en 3.1, donde se observa la descripción de una de las CPUs *Cortex A53* de *Zynq UltraScale+*.

Listado 3.1: Ejemplo de *Device Tree*.

```
/ {
  compatible = "xlnx,zynqmp";
  #address-cells = <2>;
  #size-cells = <2>;

  cpus {
    #address-cells = <1>;
    #size-cells = <0>;

    cpu0: cpu@0 {
      compatible = "arm,cortex-a53", "arm,armv8";
      device_type = "cpu";
      enable-method = "psci";
      operating-points-v2 = <&cpu_opp_table>;
      reg = <0x0>;
      cpu-idle-states = <&CPU_SLEEP_0>;
    };

  }

}
```

La compilación del DTS se realiza mediante un compilador denominado Device Tree Compiler (DTC) (integrado en el conjunto de herramientas de *Petalinux*) para generar un fichero llamado Device Tree Blob (DTB). Este último, debe ser traspasado al *kernel* de Linux por parte del *bootloader* (copiando el DTB en la memoria RAM e indicando su dirección inicial y su tamaño) durante el proceso de arranque, para identificar los recursos *hardware* de los que dispone el sistema y actuar en consecuencia cargando

los *drivers* necesarios y su configuración.

De forma genérica la sintaxis presente en un *Device Tree* incluye:

- El número y tipo de CPUs en el sistema.
- La dirección base y el tamaño en la memoria RAM.
- Los buses disponibles.
- Los distintos periféricos conectados a los buses, pudiendo indicar líneas de interrupción, relojes externos, canales asociados al DMA.

La generación del DTS que describe los componentes *hardware* de la plataforma de control y comunicaciones, se realiza de forma automática mediante *Petalinux* a partir del fichero de descripción *hardware*.

3.5.3 Espacio de usuario vs espacio del *kernel*

El sistema implementado en este TFM desarrolla la comunicación entre los *cores* disponibles en el chip *UltraScale+* tanto en el espacio de usuario como en el núcleo de *Linux* comparando ambas alternativas. Por ello, es necesario conocer las diferencias básicas entre ambos contextos a la hora de desplegar aplicaciones y otro tipo de servicios sobre un S.O.

Linux no trabaja directamente con la memoria física del sistema, sino que la mapea en direcciones virtuales proporcionando un nivel de abstracción que permite al usuario desentenderse de las direcciones físicas donde verdaderamente se ubican los dispositivos. La memoria virtual se divide en dos contextos bien diferenciados: espacio de usuario y espacio de *kernel*.

Una de las principales razones para realizar esta separación es proporcionar protección a la memoria y a los distintos periféricos, evitando el acceso a ellos por *software* malintencionado o fallos por parte del programador. La diferencia fundamental radica a nivel de privilegios, siendo el espacio de *kernel* el único capaz de ejecutar código privilegiado del S.O y acceder a los dispositivos *hardware* a través de *device drivers*.

Por otro lado, el espacio de usuario está referido a cualquier código que se ejecuta fuera del núcleo del S.O, como son las aplicaciones y servicios de los usuarios. Cada vez que un código en espacio de usuario se ejecuta se crea un proceso con un identificador asociado, Process ID (PID), el cuál dispone de su propio espacio de memoria virtual (a priori sin acceso a la memoria asignada a otra aplicación distinta) con la posibilidad de emplear llamadas al sistema o *system calls* para acceder a funcionalidades únicamente disponibles en el núcleo, como pudieran ser los *drivers* de los dispositivos o el acceso a un determinado fichero con información.

En la figura 3.27 se presenta la separación entre los contextos de usuario y núcleo del sistema operativo, donde se aprecia con claridad como el control al *hardware* subyacente es exclusivo del espacio del *kernel*.

La implementación de la comunicación entre núcleos difiere si se realiza en un espacio u otro, exponiendo los detalles y las implicaciones de ambas alternativas en el capítulo 4.

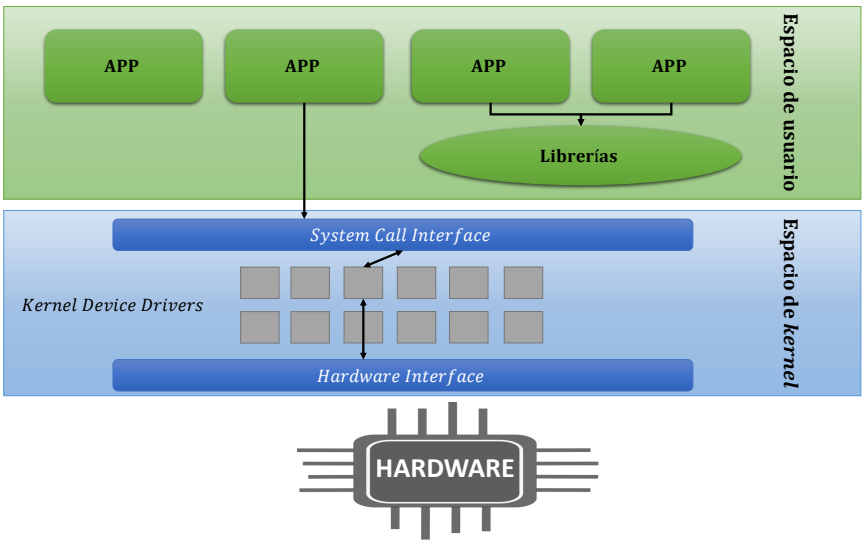


Figura 3.27: Representación gráfica de la división entre espacio de usuario y *kernel*.

3.6 Protocolo *OpenAMP*

Como se ha comentado en el apartado 1.2.1, la finalidad de este trabajo es proporcionar una plataforma flexible de control y comunicaciones para los convertidores electrónicos de potencia pertenecientes a una micro-red. El sistema se basa en la comunicación entre los contextos de la APU y RPU del chip *Zynq UltraScale+* mediante *OpenAMP*.

OpenAMP [75] es un *framework* que facilita el desarrollo de aplicaciones AMP proporcionando todos los componentes *software* necesarios. La flexibilidad de estas librerías radica en la multitud de configuraciones que pueden implementarse entre varios núcleos, como la comunicación entre un núcleo ejecutando un S.O y otro, código *baremetal*, o ambos *cores* ejecutando aplicaciones *baremetal*.

Los principales componentes *OpenAMP*, integrados en los últimos *kernels* de *Linux* son: Remoteproc, RPMsg y VirtIO.

3.6.1 *Remoteproc*

Se trata de una API compatible con la infraestructura integrada desde el *Linux Kernel 3.4.x*, que permite el control del ciclo de vida (LCM) de los procesadores remotos, proporcionando las siguientes funciones esenciales [76]:

- Permite a las aplicaciones ejecutándose en el procesador maestro cargar las secciones de código y datos del *firmware* remoto en la memoria del sistema.
- Liberar al núcleo remoto del estado de reposo para comenzar la ejecución de la aplicación remota cargada en memoria.
- Establecer el canal de comunicación RPMsg para el intercambio de información entre los contextos maestro y esclavo.
- Apagar el procesador remoto cuando sus servicios no sean necesarios.
- Proveer una API para la aplicación remota para inicializar el sistema *remoteproc* remoto y establecer la comunicación con el núcleo maestro.

El *firmware* remoto publica los recursos necesarios en una estructura de datos denominada tabla de recursos o *resource table*, donde se incluye la memoria utilizada para el código y los datos de la aplicación así como la información necesaria para implementar la comunicación RPMsg (*buffers* de memoria compartida, IPC...etc).

El núcleo maestro utiliza la información de la tabla de recursos (ubicada en una sección especial del ejecutable remoto) para reservar los recursos del sistema necesarios para la ejecución de la aplicación en el *core* esclavo.

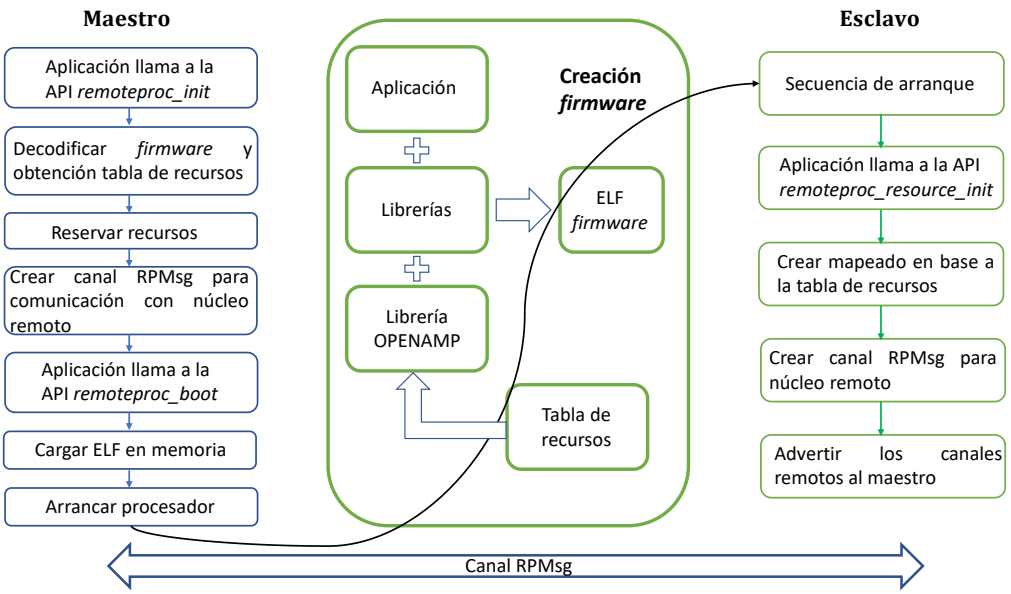


Figura 3.28: Diagrama conceptual del funcionamiento de *Remoteproc*.

En la figura 3.28 se muestra el procedimiento seguido por el componente Remoteproc para iniciar la ejecución de un código en el núcleo esclavo y establecer la comunicación entre ambos contextos. Primeramente, el maestro llama a la API *remoteproc_init* para buscar y decodificar la aplicación remota, reservando los recursos descritos en la *resource table*. Posteriormente, la llamada a *remoteproc_boot* provoca situar las secciones de código y datos del *firmware* en la memoria e iniciar la ejecución del núcleo remoto.

Una vez que la aplicación está corriendo en el procesador esclavo, se realiza una llamada a *remoteproc_resource_init* que inicializa y crea el canal de comunicación entre ambos *cores*. Finalmente, se intercambian entre ambos mensajes de aviso o *advertisement* indicando el canal RPMsg desplegado.

3.6.2 virtIO

La librería *OpenAMP* implementa el estándar virtIO para la gestión de la memoria durante la comunicación, definiendo la capa de menor nivel del *framework*. Se trata de una técnica de virtualización utilizada en S.O para los *device drivers* de discos y de red, donde solo el *driver* conoce que está siendo ejecutado en un contexto virtual.

La implementación de este estándar para la comunicación entre núcleos está basada en la gestión de la memoria compartida y en las interrupciones entre ambos *cores*. Se definen *buffers* circulares para cada sentido de transmisión en una configuración núcleo-núcleo, de forma que no es necesario ningún mecanismo de sincronización. La gestión de estos *buffers* se realiza sobre un componente denominado *vring*, formado por tres áreas bien diferenciadas (figura 3.29):

1. **Área de descripción:** empleada para detallar la ubicación y las características de los *buffers*.
2. **Área de escritura:** define listas de identificadores de *buffers* para la transmisión de información hacia el otro núcleo. Se suele denominar *available buffer*.
3. **Área de lectura:** define listas de identificadores de *buffers* para la recepción de información enviada desde el otro núcleo. Se suele denominar *used buffer*.

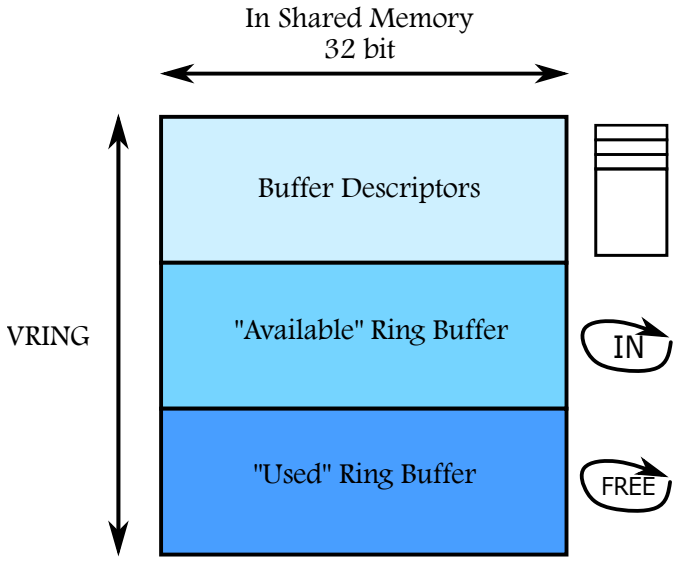


Figura 3.29: Capa virtIO OpenAMP.

3.6.2.1 Descriptores de *buffer*

El área de descripción está formada por una lista o tabla cuyos elementos se denominan descriptores de *buffer*. Cada uno de ellos contiene una dirección que indica el inicio de la información junto con un campo de longitud. Además, incluyen un campo de *flags* que modifica el acceso al *buffer* y un campo *next* que permite enlazar múltiples descriptores (figura 3.30).

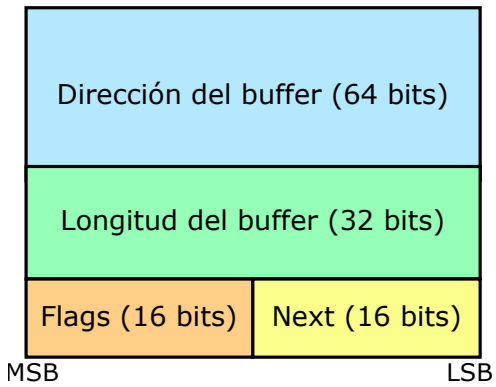


Figura 3.30: Estructura de datos de un descriptor de *buffer*.

3.6.2.2 *Available* Ring

El anillo disponible o *available ring* define una estructura de datos para la gestión de la transmisión desde uno de los extremos de la comunicación. El formato de la estructura se muestra en la figura 3.6.2.3.

El campo *idx* define un índice de cabecera que indica el siguiente *buffer* de la tabla de descriptores donde se situará la información. Por otro lado, en el campo de *flags* únicamente se emplea el bit menos significativo para indicar si se quiere interrumpir al otro *core* mediante una señal Inter-Processor Communication (IPC) para notificar la disponibilidad de nuevos datos. Por último, se dispone de una cola de índices (*ring*) que almacenan los índices de la tabla de descriptores de *buffer* que se encuentran disponibles para su lectura.

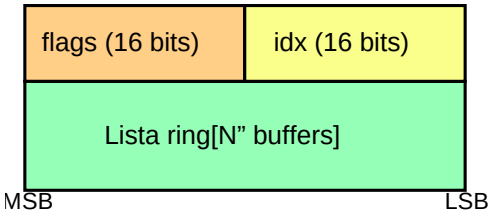


Figura 3.31: Estructura de datos de *available ring*.

Es importante remarcar que el tamaño de la lista *ring* debe tener el mismo número de elementos que descriptores de *buffer* utilizados.

3.6.2.3 *Used* Ring

El anillo usado o *used ring* tiene una estructura muy similar a la definida para el *available ring*, salvo que la lista *ring* incluye un campo adicional de longitud con el valor del tamaño total del *buffer* escrito (figura 3.32).

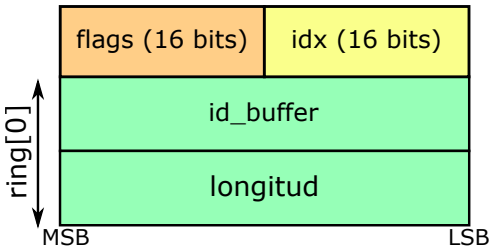


Figura 3.32: Estructura de datos de *used ring*.

En este caso, el bit menos significativo del campo *flags* permite elegir si se notifica al otro extremo cuando se modifique la estructura *used ring*.

3.6.3 RPMsg

RPMsg define un protocolo de intercambio de mensajes perteneciente a la capa de transporte en la comunicación openAMP entre los núcleos del sistema. Los mensajes RPMsg se pueden dividir en una parte de cabecera y en otra de datos útiles o *payload*, siguiendo la estructura que se muestra en la figura 3.33. Cada uno de ellos, se almacena en uno de los *buffers* reservados en la memoria compartida.

El primer campo de 32 bits corresponde a la dirección fuente, y el siguiente a la dirección destino del canal de comunicaciones. El campo reservado únicamente tiene como finalidad el alineamiento de la estructura en memoria. A continuación, el campo longitud indica el número de *bytes* de la carga útil y los *flags* se utilizan para caracterizar los datos que se encuentran al final del mensaje.

El tamaño máximo del mensaje RPMsg está definido en 512B incluyendo la cabecera, es decir, se pueden enviar hasta 496B de carga útil (512B - 16B de cabecera). Este valor se encuentra declarado

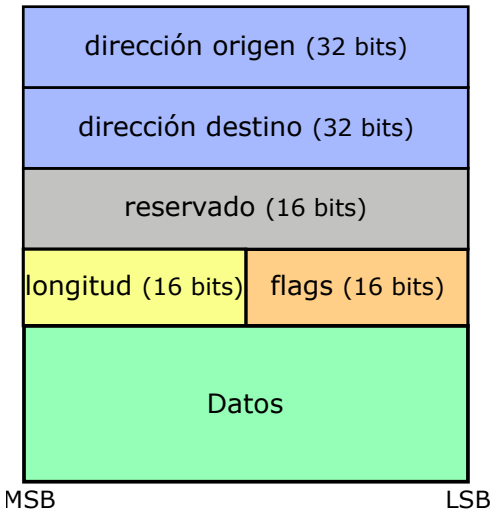


Figura 3.33: Formato mensaje RPMsg.

tanto en el núcleo de *Linux* integrado como en las librerías proporcionadas por *Xilinx*, sin embargo, se puede modificar su valor en función de las necesidades del sistema. La explicación de la modificación del tamaño máximo del mensaje se detalla en el apartado 4.

Una vez descritos todos los componentes de la comunicación OpenAMP se puede exponer el flujo que se sigue cuando el maestro envía información al núcleo remoto y viceversa. En la implementación del sistema se utilizan dos estructuras de descripción *virtIO*, VRING0 y VRING1, uno para la transmisión de información desde el maestro al esclavo y la otra para el envío de información desde el esclavo al maestro.

Por convenio, el núcleo maestro utiliza el anillo *available* para transmitir y el anillo *used* para recibir información, mientras que el remoto sigue el criterio complementario.

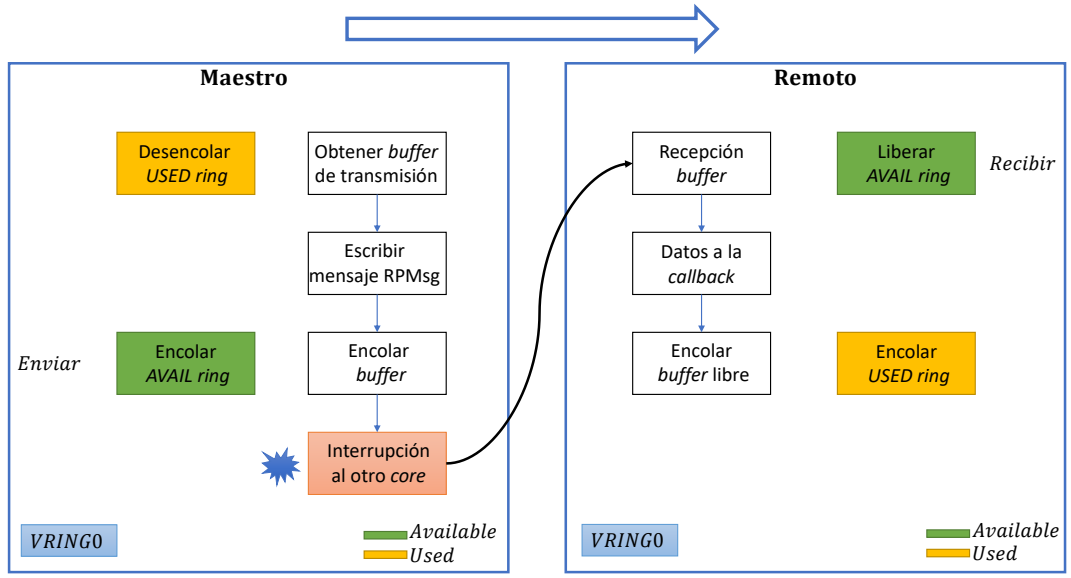
En la figura 3.34, se muestra un esquema de la comunicación en ambos sentidos, describiendo a continuación únicamente el caso de transmisión de información desde el maestro al esclavo, ya que ambas situaciones son equivalentes utilizando *vrings* distintos:

1. Se liberan del *used ring* aquellos *buffers* que hayan sido ya leídos por el otro *core*.
2. Se escribe en el *buffer* marcado por el índice de la estructura *available* el mensaje RPMsg, encolándolo en el *available ring*.
3. Se lanza una interrupción que notifica al otro núcleo la disponibilidad de nuevos datos.
4. El núcleo remoto obtiene el *buffer* y pasa los datos a la función *callback* correspondiente.
5. Se marca dicho *buffer* en*used ring*.

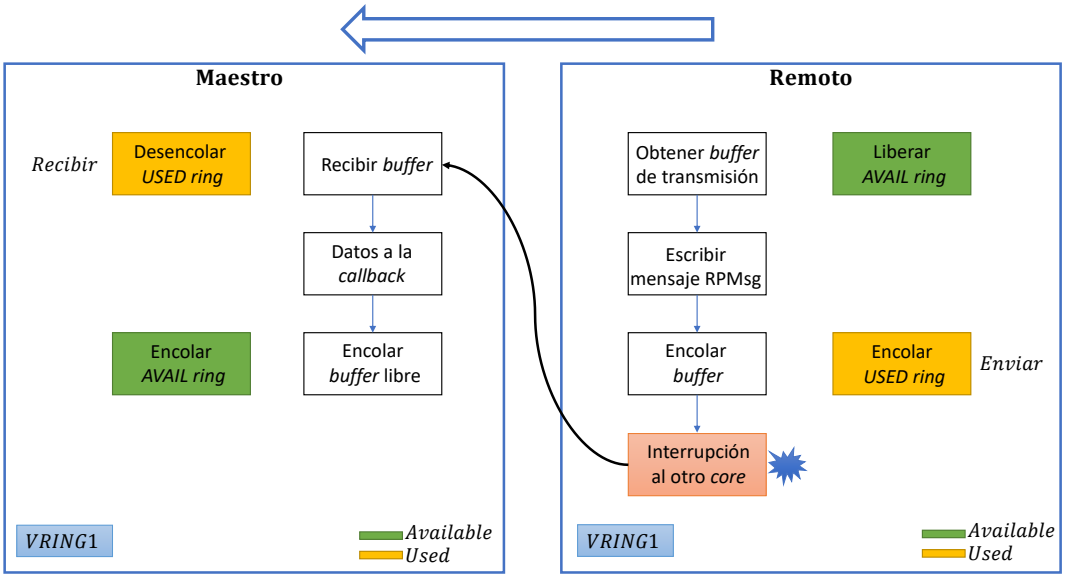
En este TFM, la comunicación siempre se realiza entre un núcleo ejecutando *Linux* como S.O y en otro núcleo código *baremetal*.

3.7 Protocolo HTTP

El protocolo de comunicación HyperText Transfer Protocol (HTTP) permite el intercambio de datos y recursos mediante el empleo de transacciones basadas en peticiones y respuestas siguiendo un modelo cliente-servidor. El uso de este protocolo se centra en el servicio de servidor *web* implementado para el



(a)



(b)

Figura 3.34: Comunicación *OpenAMP* entre: (a) Maestro → remoto y (b) Remoto → Maestro.

control de los convertidores de potencia desde cualquier dispositivo a través de un navegador. Los detalles del funcionamiento del protocolo se pueden encontrar en su correspondiente RFC [77].

En HTTP los mensajes están en texto plano (facilitando su comprensión y depuración) con la siguiente estructura:

- Línea inicial cuyo formato depende del tipo de mensaje:
  - Para las peticiones se incluye la acción o método requerido seguido de la URL del recurso y la versión HTTP del cliente.
  - Para las respuestas se incorpora primeramente la versión del protocolo seguido de un código de respuesta junto con una frase que describe de forma breve el código anterior.

- Las cabeceras del mensaje basados en caracterizar al agente usuario que solicita los recursos así como información acerca de los tipos de datos que acepta...etc.
- Cuerpo del mensaje. Es opcional y contiene los datos intercambiados entre cliente y servidor. En una petición puede incluir cierta información que quiera ser enviada al servidor, por otro lado, en una respuesta puede contener el recurso solicitado por el usuario.

En el contexto de este trabajo únicamente es necesario conocer las diferencias entre dos métodos de solicitud: GET y POST.

3.7.1 Método GET

El método de solicitud GET se emplea para pedir un determinado recurso. Generalmente, se utiliza para solicitar la visualización de páginas web, ya que los parámetros que se envían al servidor son fácilmente observables en la URL del navegador. En la figura 3.35 se muestra el formato de una solicitud de tipo GET.

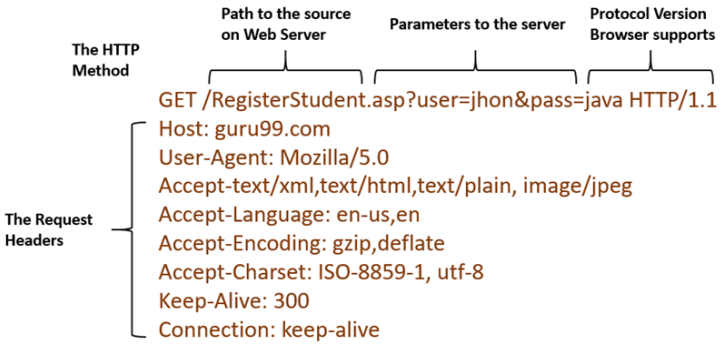


Figura 3.35: Formato método GET [15]

Los parámetros que se envían al servidor se muestran directamente en la URL del recurso solicitado, sin incluir ningún tipo de información en el cuerpo del mensaje.

3.7.2 Método POST

El método POST se emplea para el envío de parámetros así como iniciar ciertas acciones en el servidor. Los datos se incluyen en el cuerpo de la petición, es decir, no son visibles en la URL del navegador, por lo que este método es sumamente adecuado para el envío de información delicada desde un punto de vista de seguridad, ya que además, se permite la codificación de la información. En la figura 3.36 se adjunta el formato de una solicitud de tipo POST.

3.8 Protocolo *ModBus/TCP*

El protocolo *ModBus/TCP* es un servicio de comunicación basado en el intercambio de mensajes siguiendo un modelo cliente-servidor entre dispositivos conectados en una red TCP/IP ampliamente utilizado en el contexto del sector energético para el despliegue de enlaces de comunicación entre los elementos de una red eléctrica.

En este modelo se pueden diferenciar cuatro tipos de mensajes [16]:

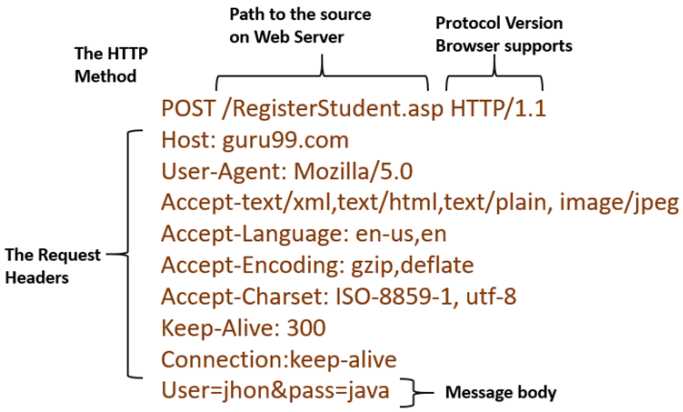


Figura 3.36: Formato método POST [15].

- Una **solicitud MODBUS** es el mensaje enviado a través de la red por el cliente para iniciar la transacción.
- Una **indicación MODBUS** es el mensaje de solicitud recibido en el lado del servidor.
- Una **respuesta MODBUS** es el mensaje de respuesta enviado por parte del servidor.
- Una **confirmación MODBUS** es el mensaje de respuesta recibido en el lado del cliente.

Cada uno de estos mensajes define una unidad de datos de aplicación (ADU) y una unidad de datos de protocolo (PDU), tal y como se muestra en la figura 3.37.

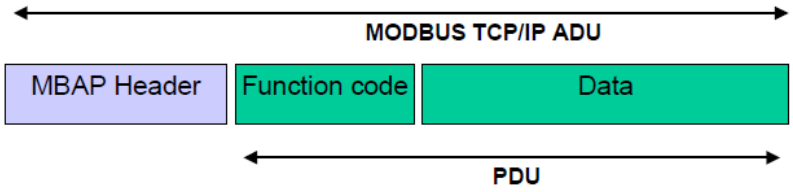


Figura 3.37: Estructura de mensajes MODBUS/TCP [16].

La cabecera MBAP (*ModBus Application Protocol*) permite identificar los mensajes MODBUS en un contexto TCP/IP. La parte de cabecera del mensaje tiene una longitud total de 7 bytes, cuyos campos junto con su descripción se detallan en la tabla 3.5.

Campos	Longitud	Descripción	Cliente	Servidor
Identificador de transacción	2 Bytes	Identificación de una transacción MODBUS	Inicializado por el cliente	Copiado por el servidor
Identificador de protocolo	2 Bytes	$0 = MODBUS$	Inicializado por el cliente	Copiado por el servidor
Longitud	2 Bytes	Número de bytes de datos	Inicializado por el cliente (solicitud)	Inicializado por el servidor (respuesta)
Identificador de unidad	1 Bytes	Identificador de un esclavo remoto	Inicializado por el cliente	Copiado por el servidor)

Tabla 3.5: Descripción campo de cabecera MBAP [16].

El código de función se describe detalladamente en [78], por lo que en este documento no se adjunta ningún tipo de descripción al respecto.

Por último, el campo de la PDU de datos está destinada a agrupar la información intercambiada entre el cliente y el servidor *ModBus*. Se debe tener en cuenta que los mensajes enviados al servidor *ModBus* deben enviarse al puerto 502.

Desde un punto de vista general, el protocolo está basado en la escritura/lectura de parámetros mapeados en memoria en el lado del servidor en forma de registro. Se definen 4 tipos de datos principales:

- ***Discrete Inputs***: 1 único bit de solo lectura.
- ***Coils***: 1 único bit de lectura/escritura.
- ***Input Registers***: registro de 16 bits de solo lectura.
- ***Holding Registers***: registro de 16 bits de lectura/escritura.

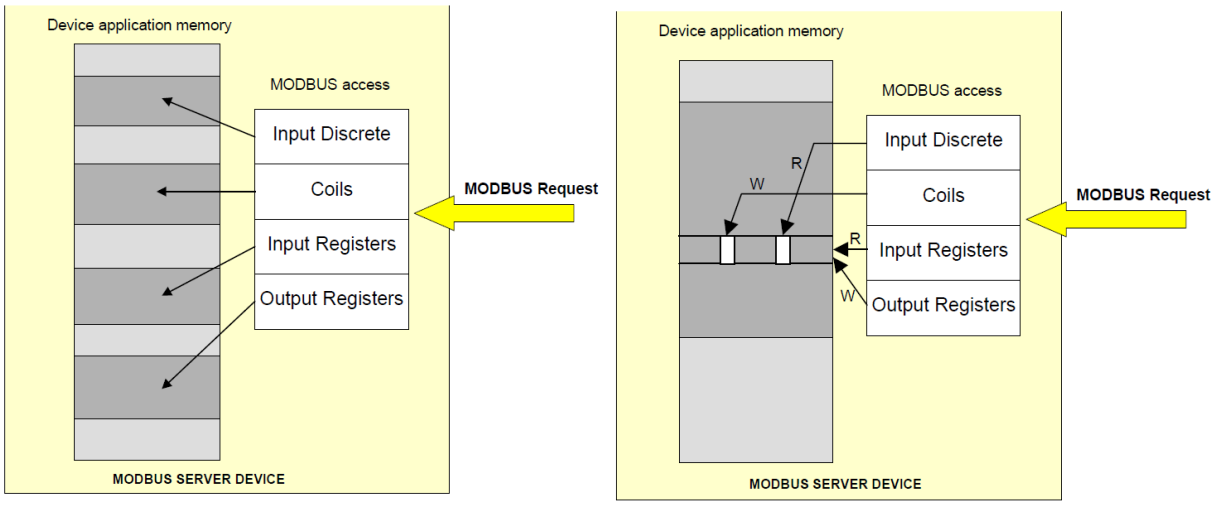


Figura 3.38: Distribución mapeado en memoria: (a) 4 áreas y (b) 1 único bloque [16].

Estos tipos de datos se pueden distribuir en 4 grandes bloques o en un único bloque, de forma que, para el primer caso, se definen áreas que contienen el mismo tipo de dato, es decir, cuando se trabaja con un mismo tipo de dato las operaciones se realizan sobre una mismo área de memoria. En contraposición, si todos los tipos de datos se disponen en un único bloque, la misma información puede ser accedida por funciones *ModBus* distintas.

En el sistema desarrollado en este TFM se ha optado por la implementación de una versión simple del protocolo *ModBus*, utilizando únicamente *holding registers* para la lectura/escritura de parámetros internos del algoritmo de control de los convertidores electrónicos de potencia de una micro-red.

3.9 Virtualización de *hardware*

El concepto de virtualización *hardware* [79] define una tecnología que introduce una capa de abstracción sobre los recursos físicos subyacentes, permitiendo la ejecución de múltiples VM de forma independiente con la creencia de disponer de todos los recursos del sistema en exclusiva para cada una de ellas. En este documento se detalla la virtualización sobre una plataforma MPSoC.

Los principales componentes de este tipo de virtualización son los siguientes [79]:

1. **Recursos *hardware***: constituyen los elementos sobre los que se aplican las técnicas de virtualización.

2. **Capa de virtualización**: determina la capa *software* de abstracción también denominada *hypervisor*.

3. **Máquinas virtuales**: es el entorno virtualizado que se ejecuta sobre su propio *hardware* virtual. Cada una de ellas están formadas por un S.O sobre el que se ejecutan múltiples aplicaciones. El número de VM está limitado por la capacidad del sistema.

El hipervisor constituye el elemento clave para aprovechar de forma eficiente todos los recursos *hardware* disponibles. Esta capa de gestión *software* fue desarrollada originalmente por los ingenieros de IBM a finales de 1960 y principios de los años 1970, cuando se integraron varias máquinas aisladas de la empresa en una sola y más grande, el *mainframe*, siendo capaz de compartir todas esos ordenadores entre distintos usuarios de forma simultánea. En función de la forma de ejecución se pueden diferenciar tres tipos de hipervisores [79]:

- **Hipervisor tipo 1**: se ejecuta directamente sobre la plataforma *hardware*, controlando los recursos físicos y los S.Os invitados. Las máquinas virtuales no tienen acceso directo al *hardware*, sino que el hipervisor actúa de intermediario para las interacciones con los recursos. Algunos ejemplos de este tipo de hipervisores son *Xen*, *Kernel-based Virtual Machine (KVM)*, *Microsoft Hyper-V*.

- **Hipervisor tipo 2**: se suele denominar *hosted*, ya que sobre el *hardware* físico se ejecuta un sistema operativo sobre el cuál actúa el hipervisor como una aplicación más del mismo proporcionando servicios de virtualización. Este tipo de hipervisor es el que se encuentra más comúnmente extendido en el mundo, siendo ejemplos aplicaciones como *VMware Workstation, VirtualBox, QEMU* y *Parallels Desktop*.

- **Hipervisor híbrido**: es una combinación de los dos anteriores, donde sobre los recursos físicos se ejecuta un sistema operativo anfitrión junto con un hipervisor, donde en ciertas ocasiones, el hipervisor puede acceder directamente al *hardware*, pero en otras necesita ciertos servicios del S.O. Como ejemplos se pueden nombrar *Microsoft Virtual Server* y *Microsoft Virtual PC*.

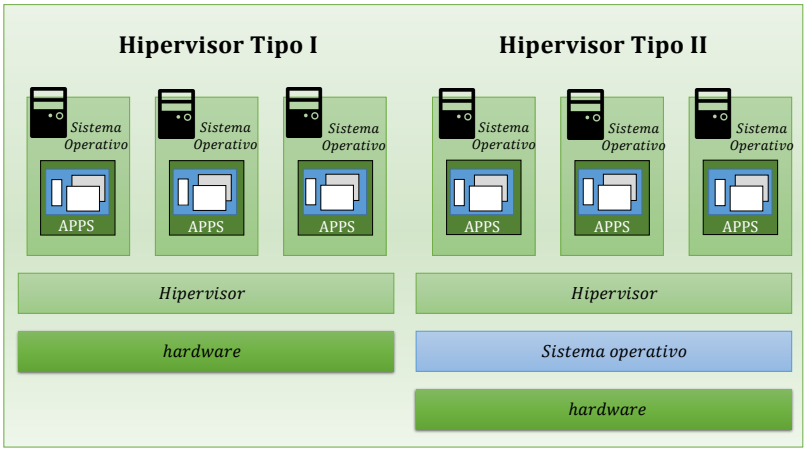


Figura 3.39: Tipos de hipervisores.

En este trabajo, la virtualización del sistema se realiza mediante un hipervisor de tipo I, concretamente, el hipervisor *Xen*.

3.9.1 Xen

Xen es un proyecto de código abierto centrado en el desarrollo de un hipervisor de tipo I que permite la ejecución de uno o varios sistemas operativos en paralelo en una misma máquina. La información detallada acerca del proyecto se encuentra en [17]. En este apartado únicamente se describen las características y conceptos fundamentales que permiten la comprensión de la virtualización de la plataforma de control y comunicaciones aplicada a las micro-redes eléctricas.

Las características principales del hipervisor *Xen* se listan a continuación [17]:

- **Interfaz y *footprint* pequeña**: ocupa alrededor de 1MB ya que está basado en un diseño de *microkernel*, con una utilización baja de memoria y una interfaz limitada con los invitados o *guests*, ofreciendo mayor robustez y seguridad.
- **Compatibilidad con varios S.O**: la gran mayoría de las implementaciones están basadas en el sistema operativo *Linux*, pero se pueden utilizar otros S.O como *NetBSD* y *OpenSolaris*.
- **Aislamiento de *drivers***: *Xen* tiene la capacidad de permitir utilizar ciertos *drivers* en un entorno virtualizado como una máquina virtual, posibilitando la recuperación ante errores sin afectar al resto del sistema.

En la figura 3.40 se muestra la arquitectura completa del hipervisor *Xen* junto con todos sus componentes, cuya descripción se adjunta seguidamente:

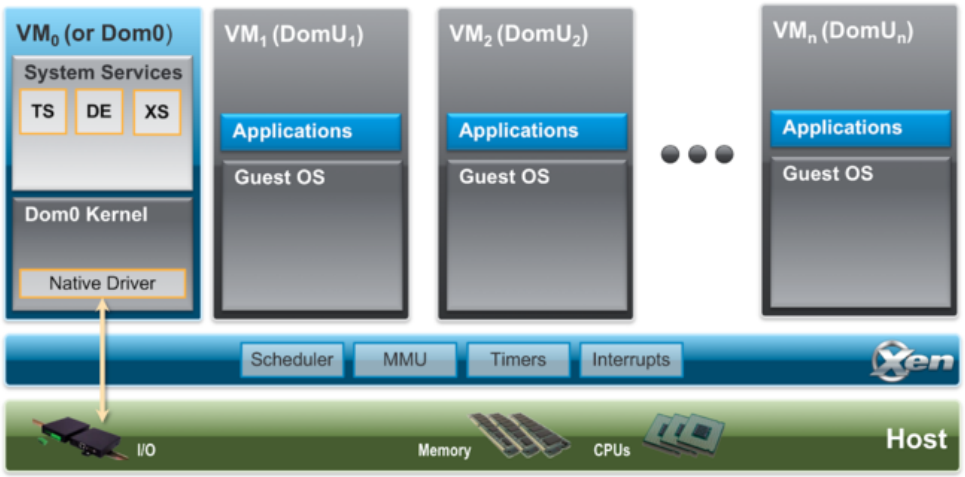


Figura 3.40: Arquitectura *Xen Project* [17].

1. El **hipervisor *Xen*** define la capa *software* de abstracción encargada de gestionar los recursos *hardware* como la CPU, la memoria, las interrupciones y los periféricos. Es el primer programa *software* que se ejecuta tras el proceso realizado por el *bootloader*.
2. Los **dominios** son entornos de virtualización que ejecutan un sistema operativo junto con sus aplicaciones. Se pueden diferenciar dos tipos de dominios:
   - **Dom0** es una máquina virtual con privilegios especiales como el acceso directo al *hardware* y la interacción con otras VM. Entre las principales funciones que posibilita este dominio están los *device drivers* físicos o virtuales (*backends*) y la creación, configuración y eliminación de instancias (DomU) dentro del sistema.

- **DomU** define un dominio virtual formado por un S.O junto con sus aplicaciones pero sin privilegios de acceso directo a los recursos físicos, sino que debe solicitar autorización al dominio privilegiado (Dom0).

Capítulo 4

Desarrollo del sistema

4.1 Introducción

El objetivo fundamental de este TFM es la implementación de un sistema de control y comunicaciones para los convertidores electrónicos de potencia integrados en una micro-red eléctrica, tal y como se plantea en el apartado 1. La plataforma está desarrollada en la última tecnología de MPSoC (*Zynq UltraScale+*), combinando los contextos de procesamiento (APU y RPU) y de lógica programable.

La APU se destina a la ejecución de un S.O como *Linux* para la gestión de las comunicaciones con el exterior y el despliegue de servicios como un servidor web y *ModBus* que permitan la visualización y el control del estado del convertidor así como la transmisión de referencias y recogida de datos en tiempo real.

Por otro lado, la RPU se emplea para los algoritmos de control de nivel primario, dada su alta exigencia computacional por los tiempos de muestreo de las señales necesarias para la implementación de los controladores así como la frecuencia de control para la conmutación de los dispositivos de potencia.

La PL se utiliza para la combinación de múltiples Intellectual Properties (IPs) desarrolladas por el grupo de investigación (GEISER) que realizan la gestión de la placa interfaz incluyendo los elementos necesarios para adquirir las señales de tensión y corriente en distintos puntos del convertidor (ADCs), las señales de actuación sobre los dispositivos de conmutación y sobre los contactores que permiten la correcta operación del convertidor.

En la figura 4.1 se muestra en detalle la arquitectura del sistema propuesto, donde se observan las partes fundamentales de la plataforma y la distribución de los distintos elementos.

La comunicación entre los núcleos pertenecientes a la APU con los de la RPU se realiza mediante las librerías *OpenAMP* de dos formas bien diferenciadas, en el espacio del *kernel* del S.O mediante el uso de un *device driver* y en el espacio de usuario. Así mismo los servicios de comunicaciones en *Linux* (servidor web y *ModBus*) se han planteado siguiendo dos posibles arquitecturas que se describen en los sucesivos apartados: arquitectura monolítica y arquitectura basada en microservicios.

La integración de técnicas de virtualización sobre la plataforma tiene la finalidad de proporcionar una mayor flexibilidad y versatilidad al sistema, ya que sin ella, el uso de *Linux* se plantea siguiendo una arquitectura SMP con el control sobre los 4 *cores* A-53 de la APU sin poder decidir qué aplicación se ejecuta en cada núcleo en cada instante (la responsabilidad de seleccionar a qué proceso se asigna recursos de procesamiento y durante cuánto tiempo recae sobre el planificador del S.O). La virtualización posibilita la creación y despliegue de máquinas virtuales ejecutándose sobre CPUs virtuales, pudiendo

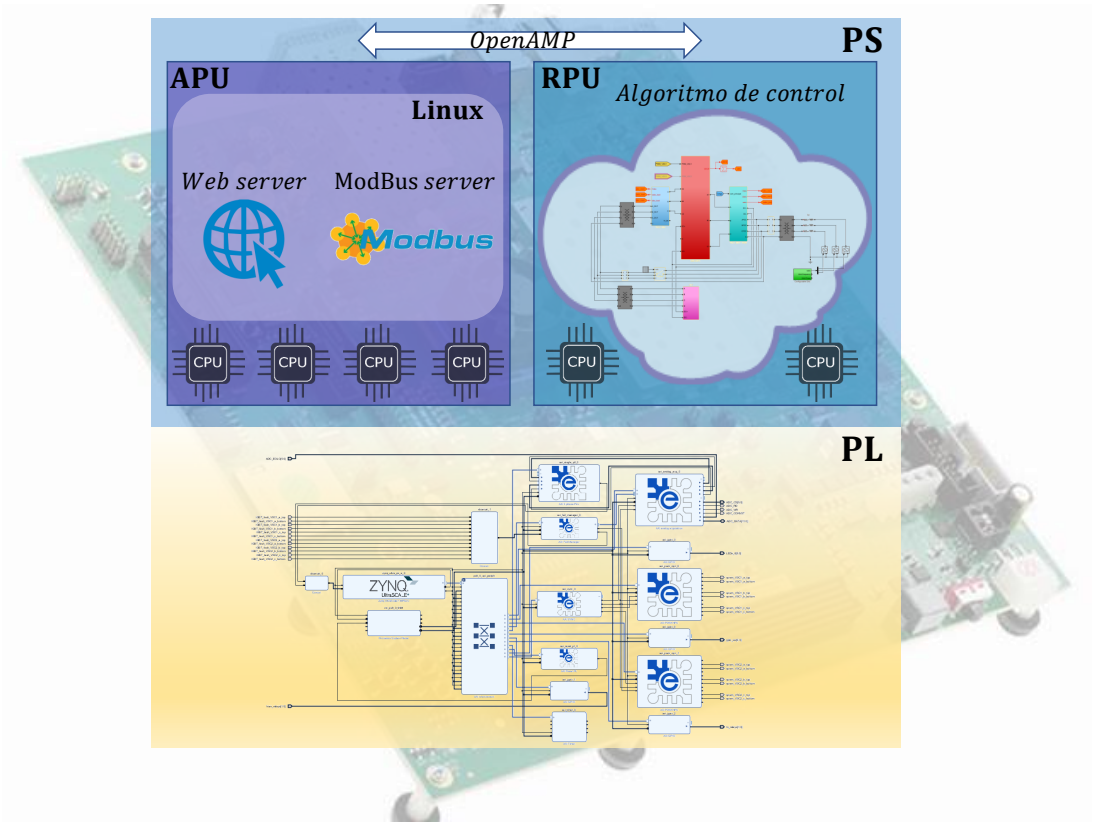


Figura 4.1: Infraestructura del sistema de control y comunicaciones propuesto.

decidir en qué CPU física concreta realizar el despliegue. En la figura 4.2 se adjunta la arquitectura del sistema utilizando técnicas de virtualización mediante el uso del hipervisor *Xen*. Se debe apreciar como la única modificación con respecto al sistema sin virtualización se encuentra en el contexto de la APU, donde es el hipervisor el que maneja el *hardware* con el dominio de control *Dom0* y despliega dominios sin privilegios *DomU*.

La explicación de la plataforma se centra en la integración y despliegue de los distintos elementos del sistema, así como la interacción entre todos ellos. Los detalles de la implementación de los algoritmos de control utilizados sobre el convertidor electrónico de potencia que se describe en la sección 5 no se incluyen en este documento. De igual forma, no se adjunta la explicación del diseño, configuración y utilización de las IPs desarrolladas por el grupo de investigación GEISER.

La exposición de la integración de *Linux*, los servicios de servidor web y *ModBus*, así como la comunicación entre *cores* (incluyendo diferentes arquitecturas) y la configuración de la PL y la plataforma completa se expone a lo largo de este capítulo.

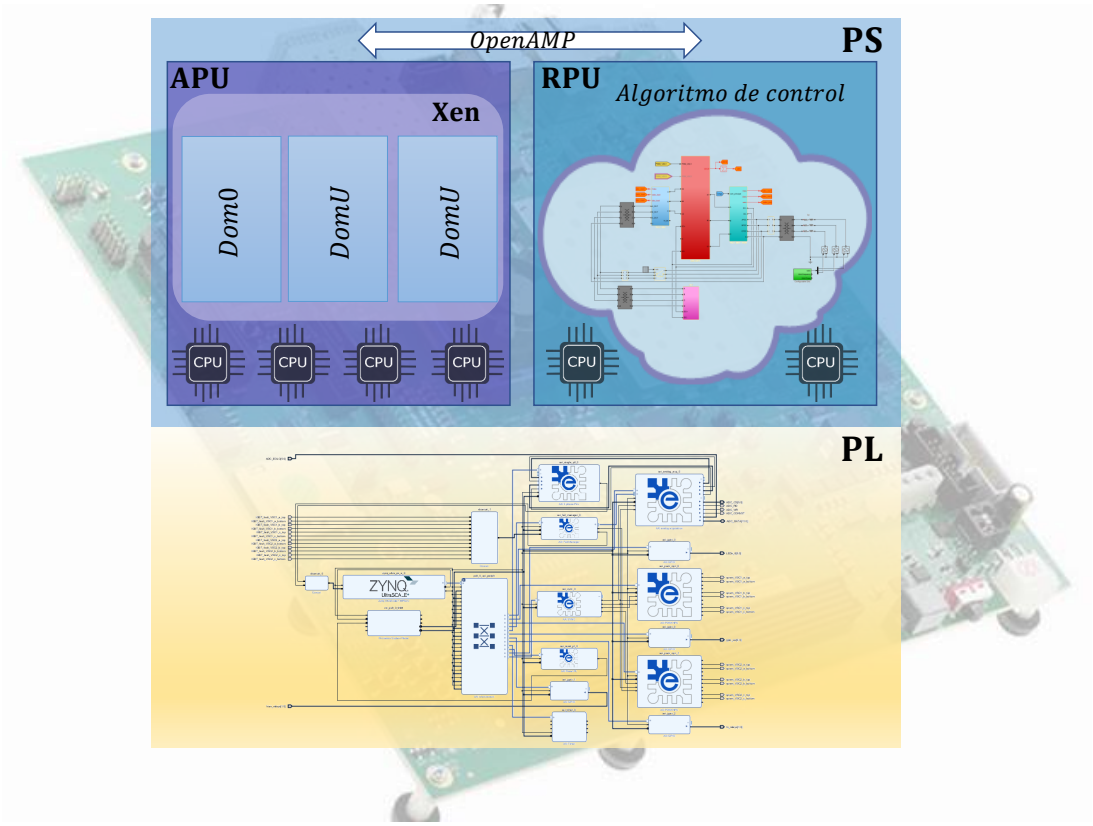


Figura 4.2: Infraestructura del sistema de control y comunicaciones mediante virtualización.

4.2 Implementación *Linux* en MPSoC

La integración de *Linux* sobre el MPSoC *Zynq UltraScale+* se ha realizado con el conjunto de herramientas de *Petalinux*. Primeramente, tal y como se expuso en 3.5.1, se debe partir de un fichero de descripción *hardware* (hdf) que permita conocer al S.O todos los recursos físicos de los que dispone.

Petalinux ofrece la posibilidad de generar los ficheros necesarios para la configuración automática de la FPGA en el momento del arranque del sistema, sin embargo, la propuesta incluida en este documento realiza la carga de la PL en caliente, una vez *Linux* se encuentra operativo a través del *framework FPGA Manager*, descrito en apartados posteriores. La razón principal de optar por una configuración en caliente es para reducir el tiempo de desarrollo de la plataforma e independizarla de *Petalinux*, ya que si se carga la FPGA al arrancar el sistema, implica que ante un cambio en el diseño *hardware* en alguna IP, es necesario volver a generar nuevos ficheros a través de *Petalinux*. En contraposición, el empleo de *FPGA Manager* permite generar el fichero *bitstream* y cargarlo de forma *online* sin necesidad de volver a recompilar el núcleo del S.O ni apagar la placa de desarrollo.

Por todo ello, el diseño parte de la IP que representa la PS de *Zynq UltraScale+* junto con su correspondiente sistema de *reset*, tal y como se adjunta en la figura 4.3. De esta forma, se consigue disponer de un diseño base que contiene la descripción de todos los recursos *hardware* correspondientes al sistema de procesamiento para la creación de la imagen de *Linux*. Este proceso solo se realiza una única vez ya que la configuración de la PL se realiza mediante el *framework FPGA Manager*.

Una vez realizado el proceso de síntesis e implementación en *Vivado* se genera el fichero de descripción

Figura 4.3: Bloque de diseño base para la integración de *Linux* sobre la plataforma.

hardware .hdf a partir del cual se crean las imágenes de *Linux* con el *software* Petalinux.

El primer paso con Petalinux es la creación de un proyecto con el comando *"petalinux-create"* seleccionando la plantilla *hardware* de *Zynq UltraScale+* junto con el nombre que se desee dar al proyecto:

```
$ petalinux-create --type project --template zynqmp --name <nombre_proyecto>
```

La elección de la plantilla (*template*) en el comando anterior, permite generar un fichero DTS que incluye todos los periféricos disponibles en la plataforma *hardware*. Posteriormente, se utiliza el comando *"petalinux-config"* para personalizar el proyecto con el diseño concreto desarrollado (figura 4.3):

```
$ petalinux-config --get-hw-description=<Path fichero HDF>
```

La inclusión del fichero de descripción *hardware* en *Petalinux* permite compararlo con la plantilla base disponible y habilitar/deshabilitar los periféricos que se han seleccionado en el bloque de diseño de *Vivado*. Seguidamente, se configura la fuente de arranque del sistema así como la ubicación del sistema de ficheros de *Linux* mediante el comando *"petalinux-config"*:

```
$ petalinux-config
```

En la interfaz gráfica se debe seleccionar como unidad de arranque una tarjeta SD y como ubicación del sistema de ficheros la opción *INITRAMFS*, que permite la carga en memoria RAM de la estructura de ficheros de *Linux* en el arranque. Lo anterior permite acceder al contenido de forma muy rápida pero tiene la desventaja de ser un sistema temporal, ya que al apagar la placa de desarrollo, los cambios realizados no se guardan de forma permanente.

Petalinux proporciona la posibilidad de crear aplicaciones dentro de un proyecto mediante el comando *"petalinux-create"*:

```
$ petalinux-create --type apps --name <nombre_app>
```

En la imagen de *Linux* utilizada en el desarrollo de este sistema, se ha creado una aplicación que actúa como *bootscript*, es decir, se trata de un programa que se ejecuta de forma automática al inicio, una vez el núcleo y el sistema de ficheros se encuentran operativos. En 4.1 se adjunta el código correspondiente al *bootscript* donde se realiza el montaje de las particiones BOOT y ROOTFS de la tarjeta SD, se crea el directorio *"/lib/firmware"* para alojar las aplicaciones de control que se ejecutarán sobre los núcleos R5 de tiempo real y se instala el *device driver "rpmsg_user_dev_driver"*, cuya descripción se expone en el apartado correspondiente a la implementación de OpenAMP en el espacio de *kernel*. Finalmente, se ejecuta un *script* denominado *init.sh* para la configuración y despliegue inicial de la plataforma.

Listado 4.1: Código *bootscript*.

```
#!/bin/sh

set -e
echo "### Bootscript v5 ### Start."

# It is useful to have access to the BOOT partition in order to update the Linux
# image without removing the SD card, you only need to replace the BOOT.bin and
# image.ub files. To access this partition it is necessary to mount it in a
# local Linux folder.
mkdir /home/root/boot

# The operating system does not save the changes established in last Linux
# session. For this reason, the ROOTFS partition is used. Each time Linux is
# booted, it mounts that partition to a local folder and makes a copy of it in
# another directory. This technique is very effective to extend the SD card life
mkdir /home/root/rootfs #used by temp files
mkdir /home/root/sd_rootfs #used by fixed files

# Mount "BOOT" partition of SD card into "/home/root/boot"
mount /dev/mmcblk0p1 /home/root/boot
chmod -R 755 /home/root/boot

# Mount "ROOTFS" partition of SD card into "/home/root/sd_rootfs"
mount /dev/mmcblk0p2 /home/root/sd_rootfs
chmod -R 755 /home/root/sd_rootfs

# Copy content (sw, bitfile, drivers...) of "sd_rootfs" into local rootfs folder
cp -R /home/root/sd_rootfs/* /home/root/rootfs
chmod -R 755 /home/root/rootfs

# Create /lib/firmware
mkdir /lib/firmware

# Load rpmsg user module
modprobe rpmsg_user_dev_driver

# Init Scripts
cd /home/root/rootfs/scripts
./init.sh

echo "### Bootscript v5 ### End successful."
```

Para incluir la aplicación *bootscript* en la imagen de *Linux* así como la instalación de paquetes básicos que dispone cualquier S.O moderno se utiliza el siguiente comando:

```
$ petalinux-config -c rootfs
```

En el menú desplegable se debe habilitar la aplicación *bootscript* generada para que Petalinux la compile junto con todo el núcleo de *Linux* y la incluya en el sistema de ficheros:

```
    apps --->
        [*] bootscript
```

Como último paso de personalización de la imagen de *Linux* se deben activar una serie de opciones en el núcleo del S.O con el siguiente comando:

```
$ petalinux-config -c kernel
```

Para trabajar con *OpenAMP* se deben habilitar las siguientes opciones en el menú desplegable. Todas ellas se describen en [75], pero se vuelven a incluir aquí para facilitar la comprensión del lector:

```
        [*] Enable loadable module support --->

        Device Drivers --->
                Generic Driver Options --->
                        <*> Userspace firmware loading support

        Device Drivers --->
                Remoteproc drivers --->
                        <M> ZynqMP_r5 remoteproc support
```

Posteriormente, se debe modificar el *Device Tree* generado por defecto con el fichero *system_user.dtsi* (generado sin contenido de forma automática por *Petalinux*) para incluir información relativa a *Remoteproc* y la ubicación de los *descriptores de buffer* y la memoria compartida entre núcleos. La descripción del contenido de este fichero se realiza en los apartados 4.4 y 4.3, ya que son radicalmente diferentes.

La orden que permite realizar una compilación completa del núcleo y el sistema de ficheros de *Linux* es la siguiente:

```
$ petalinux-build
```

El proceso de compilación es largo en términos temporales por lo que requiere una elevada potencia computacional si se desea finalizarlo de forma rápida.

Por último, se generan los ficheros que se deben incluir en la partición primaria de la tarjeta SD para ejecutar el S.O sobre el MPSoC, *BOOT.bin* e *image.ub* mediante el siguiente comando:

```
$ petalinux-package --boot --fsbl images/linux/zynqmp_fsbl.elf --u-boot --force
```

El contenido de los archivos *BOOT.bin* e *image.ub* se describe en detalle en la tabla 4.1.

Fichero	Componente	Descripción
BOOT.bin	zynqmp_fsbl	First Stage Boot Loader (FSBL) configura la FPGA si se dispone de un archivo *bitstream* y se encarga de cargar el Second Stage Boot Loader (SSBL).
	U-boot	Se trata del *bootloader* a partir del cual se interpreta el fichero *image.ub* para desplegar el núcleo y el sistema de ficheros de *Linux*.
	pmufw	*Firmware* destinado para extender la funcionalidad de la PMU. Se genera por defecto pot *Petalinux*.
	bl31	Componente creado de forma automática por *Petalinux* que proporciona una referencia para la ejecución en modo seguro de código (incluyendo al S.O). Se lanza en el nivel de excepción más alto (EL3).
image.ub	system.dtb	*Device Tree* que especifica los recursos *hardware* de los que dispone *Linux*.
	Image	Imagen del núcleo de *Linux*.
	rootfs.cpio.gz	Sistema de ficheros comprimido.

Tabla 4.1: Componentes internos de los ficheros BOOT.bin e image.ub.

4.2.1 Preparación de la tarjeta SD

El arranque del sistema se realiza a través de una tarjeta SD que debe incluir dos particiones: BOOT y ROOTFS. La primera corresponde a la partición de arranque que contiene los ficheros *BOOT.bin* e *image.ub* junto con una estructura de directorios como la que se muestra en la figura 4.4. La partición ROOTFS se emplea para almacenar los *scripts* de gestión de la plataforma y los registros o *loggings* con información relevante acerca de las pruebas que se realicen.

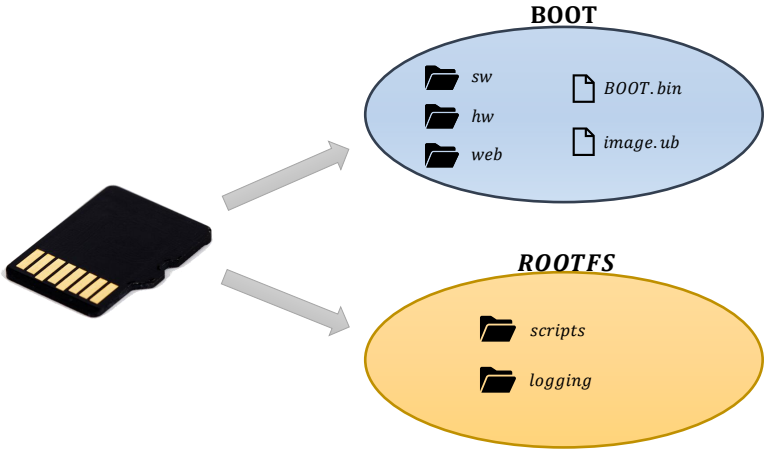


Figura 4.4: Particiones tarjeta SD.

Las características de cada partición se resumen a continuación:

1. **Partición BOOT:**
   - Tamaño: 1GB
   - Tipo: Primaria
   - Formato: FAT32
   - Nombre: BOOT
2. **Partición ROOTFS:**
   - Tamaño: espacio restante disponible
   - Tipo: Primaria
   - Formato: FAT32
   - Nombre: ROOTFS

Las particiones están organizadas siguiendo una estructura de carpetas que define una jerarquía ordenada para almacenar los elementos necesarios para el despliegue de la plataforma. La descripción de cada una de ellas se expone seguidamente:

- **sw**: en esta carpeta se almacenan los códigos que implementan los servicios de comunicación con el exterior (servidor web y servidor *ModBus*) así como el código *baremetal* que incluye los algoritmos de control de los convertidores.
- **hw**: incluye los ficheros de tipo *bitstream* que permiten la configuración de la FPGA.

- **web**: en este directorio se guardan las páginas web y las imágenes utilizadas por parte del servidor web.
- **scripts**: incluye todos los *scripts* de gestión y control del sistema desarrollado.
- **logging**: se almacenan los registros de datos realizados durante el funcionamiento de la plataforma.

4.2.2 *FPGA Manager*

El *framework FPGA Manager* se ha utilizado para la programación *online* de la lógica programable a través del S.O *Linux* (figura 4.5). Proporciona un *device driver* con la capacidad de realizar una carga completa o parcial de la FPGA, con ficheros *bitstream* seguros (encriptados) o no seguros con la posibilidad de incorporar autenticación.

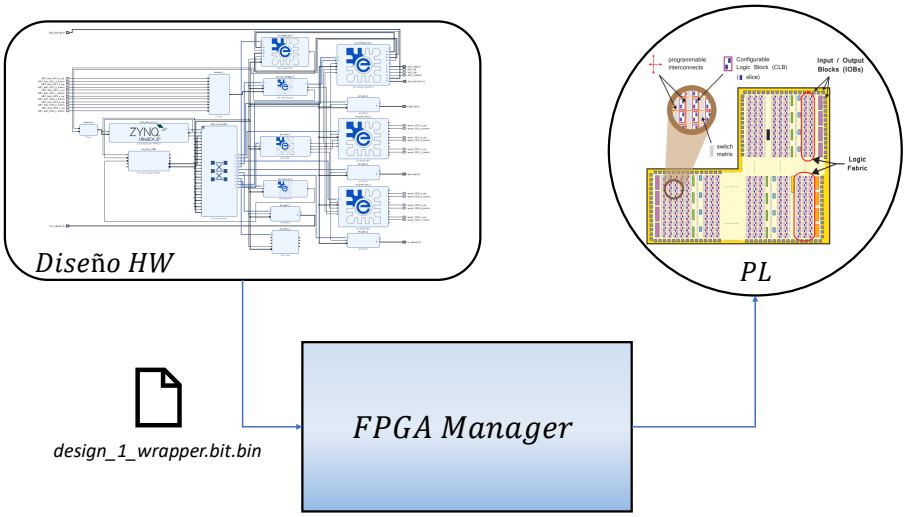


Figura 4.5: Flujo de configuración de la PL mediante el *framework FPGA Manager*.

En la creación de la imagen de *Linux*, la opción para incorporar este *framework* de manejo de la FPGA se encuentra activado por defecto en *Petalinux*. Igualmente, la opción que debe marcarse se adjunta a continuación en el menú de configuración del núcleo:

```
Device Drivers --->
        FPGA Configuration Framework --->
                <*> Xilinx ZynqMP FPGA
```

El desarrollo de este trabajo se ha realizado con la versión *Vivado 2018.2*, lo que supone ciertas particularidades en cuanto al formato del fichero *bitstream*. El *framework* en esta versión únicamente admite formatos con la extensión ".bin", por lo que es necesario el uso de un comando del programa de desarrollo *software* integrado en *Vivado*, SDK, para transformar el fichero *bitstream* con extensión ".bit" generado en el flujo de síntesis e implementación de la parte lógica de la plataforma. El comando se adjunta a continuación:

```
bootgen -image <fichero .bif> -arch zynqmp -o ./<fichero de salida .bin> -w
```

El fichero .bif especifica los elementos que se van a utilizar para generar el archivo de salida, así como su componente destino. En este caso, tal y como se muestra en el código 4.2, se indica la ruta donde se encuentra el fichero .bit de entrada (*design_1_wrapper.bit*) y el dispositivo destino, la PL.

Listado 4.2: Código fichero .bif.

```
all:
{
        [destination_device = pl] .\design_1_wrapper.bit /* Bitstream file name */
}
```

El fichero que describe el diseño de la lógica programable en formato binario (.bin) es el que se emplea como entrada al *framework FPGA Manager* para realizar la configuración de la PL.

La carga de la FPGA se realiza con los siguientes comandos en la *shell* de *Linux*:

```
echo 0 > /sys/class/fpga_manager/fpga0/flags
echo design_1_wrapper.bit.bin > /sys/class/fpga_manager/fpga0/firmware
```

El primero de ellos, se utiliza para indicar una configuración completa de la FPGA, también denominada *Full Bitstream*. El segundo introduce el fichero binario como *firmware* del *device driver* y realiza el proceso de escritura en la PL.

4.3 OpenAMP en espacio de *kernel*

La implementación de la comunicación *OpenAMP* en el espacio del *kernel* de *Linux* está basada en el empleo de *Remoteproc* y un *driver* del bus RPMsg (*rpsmg_user_dev_driver*) que proporciona una interfaz de acceso para las aplicaciones desde el espacio de usuario.

En esta propuesta se debe compilar el núcleo con el *device tree* que se muestra en el código 4.3. En él se observa la reserva de una zona de memoria destinada al componente *Remoteproc* para que ubique el *firmware* correspondiente a los algoritmos de control, así como la memoria compartida entre los contextos APU y RPU. La dirección base de la memoria reservada es *0x3ed00000* con un tamaño de *0x400000*, utilizando la palabra clave *"no-map"* para indicar que esa región de memoria no debe ser mapeada por el S.O (línea 10).

Además, es necesario declarar los dominios de potencia (*power-domains* correspondientes a los núcleos, en este caso 1 único núcleo R5 (líneas 15-18) y a las memorias propias TCM-A y TCM-B que se van a emplear.

Por último, en el nodo AMBA se especifican las direcciones base y sus correspondientes tamaños de las memorias disponibles (DDR, TCM-A y TCM-B) para ubicar las distintas partes del código *baremetal* de control y el componente *Remoteproc* (*zynqmp_r5_rproc@0*). En este último se declaran los registros de control (línea 45-48) de los núcleos R5 de tiempo real así como la línea de interrupción entre núcleos (IPI). El modo de funcionamiento de la RPU se configura como *split* (línea 50), es decir, los *cores* se ejecutan de forma independiente. En la verificación del sistema desarrollado únicamente se ha empleado un núcleo R5.

En el desarrollo del código *baremetal* que incluye los distintos algoritmos de control primario de los convertidores, se genera un fichero de forma automática denominado *linker script*. Este archivo describe la ubicación de las distintas secciones del ejecutable generado, pudiendo controlar en qué recursos de memoria deben situarse dichas secciones de código. En la figura 4.6 se refleja el *linker script* correspondiente al ejecutable generado en formato Executable and Linkable Format (ELF), donde se observan los mismos recursos de memoria especificados en el *device tree*.

Listado 4.3: *Device Tree* para la implementación de *OpenAMP* en el espacio del núcleo.

```
/include/ "system-conf.dtsi"

/ {
  reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;
    /* Reserved DDR memory for RPU firmware and shared memory between APU and RPU */
    rproc_0_reserved: rproc@3ed00000 {
      no-map;
      reg = <0x0 0x3ed00000 0x0 0x400000>;
    };
  };
  power-domains {
    pd_r5_0: pd_r5_0 {
      #power-domain-cells = <0x0>;
      pd-id = <0x7>;
    };
    pd_tcm_0_a: pd_tcm_0_a {
      #power-domain-cells = <0x0>;
      pd-id = <0xf>;
    };
    pd_tcm_0_b: pd_tcm_0_b {
      #power-domain-cells = <0x0>;
      pd-id = <0x10>;
    };
  };
amba {
  r5_0_tcm_a: tcm@ffe00000 {
    compatible = "mmio-sram";
    reg = <0x0 0xFFE00000 0x0 0x10000>;
    pd-handle = <&pd_tcm_0_a>;
  };
  r5_0_tcm_b: tcm@ffe20000 {
    compatible = "mmio-sram";
    reg = <0x0 0xFFE20000 0x0 0x10000>;
    pd-handle = <&pd_tcm_0_b>;
  };
  elf_ddr_0: ddr@3ed00000 {
    compatible = "mmio-sram";
    reg = <0x0 0x3ed00000 0x0 0x40000>;
  };
 test_r5_0: zynqmp_r5_rproc@0 {
   compatible = "xlnx,zynqmp-r5-remoteproc-1.0";
   reg = <0x0 0xff9a0100 0x0 0x100>,
         <0x0 0xff340000 0x0 0x100>,
         <0x0 0xff9a0000 0x0 0x100>;
   reg-names = "rpu_base", "ipi", "rpu_glbl_base";
   dma-ranges;
   core_conf = "split0";
   srams = <&r5_0_tcm_a &r5_0_tcm_b &elf_ddr_0>;
   pd-handle = <&pd_r5_0>;
   interrupt-parent = <&gic>;
   interrupts = <0 29 4>;
 };
};

};
```

Es sumamente importante que exista una correspondencia total entre las direcciones base y los tamaños asignados a cada uno de los recursos de memoria descritos en el *device tree* y en el *linker script* para que el componente *Remoteproc* sitúe el código *baremetal* de forma correcta en memoria, sin acceder a ninguna dirección que tenga mapeada el sistema operativo.

De forma similar a los comandos introducidos en el *framework FPGA Manager* para la configuración de la PL, se utilizan las siguientes interfaces proporcionadas por *Remoteproc* para controlar el ciclo de

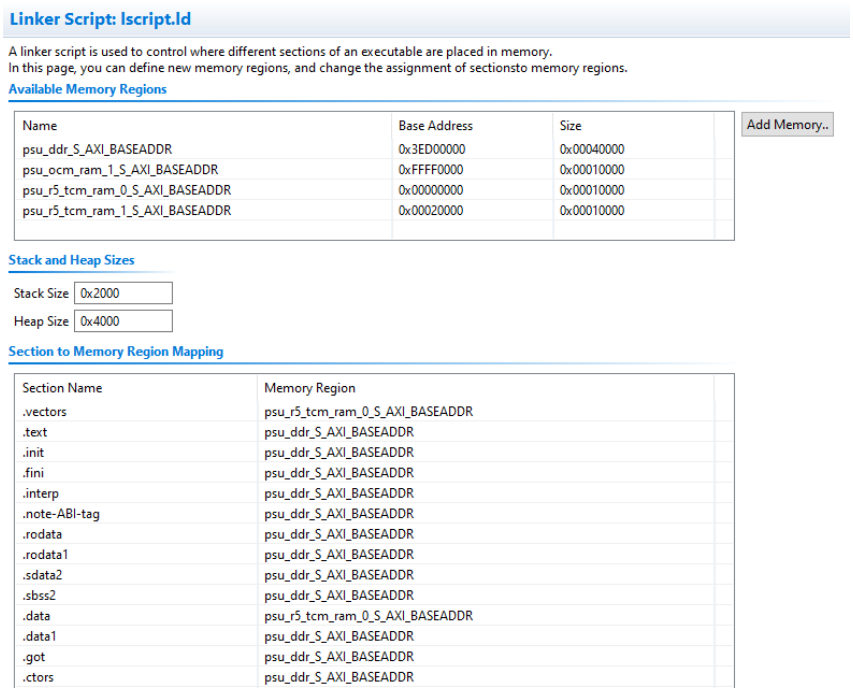


Figura 4.6: *Linker script* correspondiente al código *baremetal* de control.

vida de los núcleos R5 de la RPU:

```
echo <ELF code> > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
echo stop > /sys/class/remoteproc/remoteproc0/state
```

La primera orden indica al componente *Remoteproc* el programa ELF que debe cargar en memoria e iniciar su ejecución en uno de los procesadores de tiempo real. Posteriormente, los comandos de *"start"* y *"stop"* permiten el control del ciclo de vida del procesador, iniciando la ejecución del *firmware* y apagando/liberando los recursos asignados respectivamente.

Una vez se inicializa *Remoteproc* al estado de *"start"* se llevan a cabo las acciones que se representaron en la figura 3.28, generando las relaciones entre ficheros y *device drivers* que se adjunta en la figura 4.7. En ella, se observa el bus *RPMsg* (creado a partir de las características descritas en el *RPMsg Core*) que concentra la información acerca de los dispositivos de comunicación *RPMsg* así como los *device drivers* que los manejan.

En concreto, un elemento fundamental en la implementación de *OpenAMP* en el espacio del *kernel* de *Linux* es el *driver "rpmsg_user_dev_driver"*, el cuál es responsable de actuar de interfaz entre el canal de comunicación entre *cores* y la aplicación corriendo en el espacio de usuario del S.O. El acceso al mismo se realiza a través de llamadas al sistema como *open*, *close*, *read* y *write* sobre un fichero denominado */dev/rpmsg0* que representa el canal de comunicaciones *RPMsg* creado.

El intercambio de información entre los núcleos mediante el empleo del *driver "rpmsg_user_dev_driver"* se representa en la figura 4.8. Los servicios de comunicación (servidor web y ModBus) acceden al canal *RPMsg* a través del fichero */dev/rpmsg0* leyendo y escribiendo información mediante las llamadas al sistema *read* y *write* respectivamente. El *driver* es el encargado de escribir el contenido de los datos en el formato correcto, es decir, añadiendo los campos de cabecera de un mensaje *RPMsg*, en el *buffer* reservado para ese fin así como iniciar la Inter-Processor Interrupt (IPI) que notifique al otro núcleo la disponibilidad de nueva información.

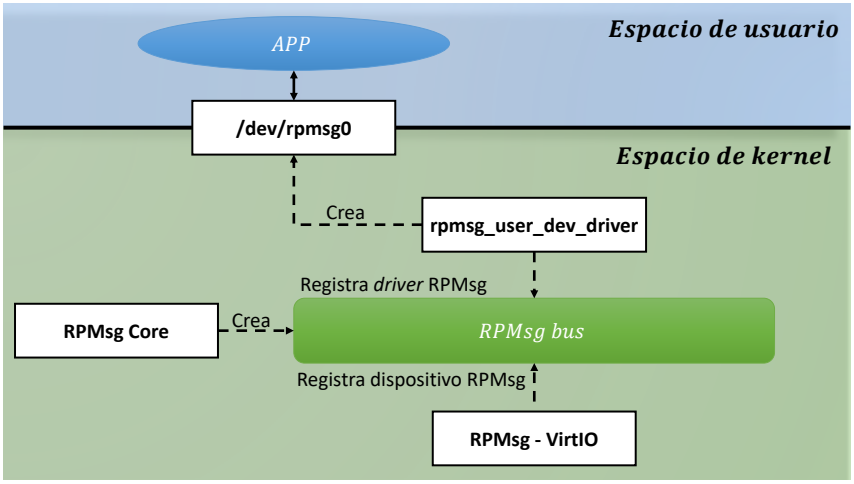


Figura 4.7: Estructura comunicación *OpenAMP* en el espacio del *kernel* de *Linux*.

En el sentido contrario es similar, una vez el código *baremetal* envía nuevos datos utilizando la API *RPMsg* disponible en la librería *OpenAMP*, se lanza una IPI hacia el otro *core*, ejecutando la *callback* correspondiente en el *driver*. En ella, básicamente se decodifican los datos y se pasan desde el espacio de *kernel* hasta el espacio de usuario para que las aplicaciones puedan utilizar esa información.

La ubicación de los *buffers* que se describen en las estructuras *vring* se realiza de forma dinámica por el S.O a través de Contiguous Memory Allocator (CMA), es decir, cada vez que la plataforma se inicia, la disposición en memoria de dichos *buffers* se modifica.

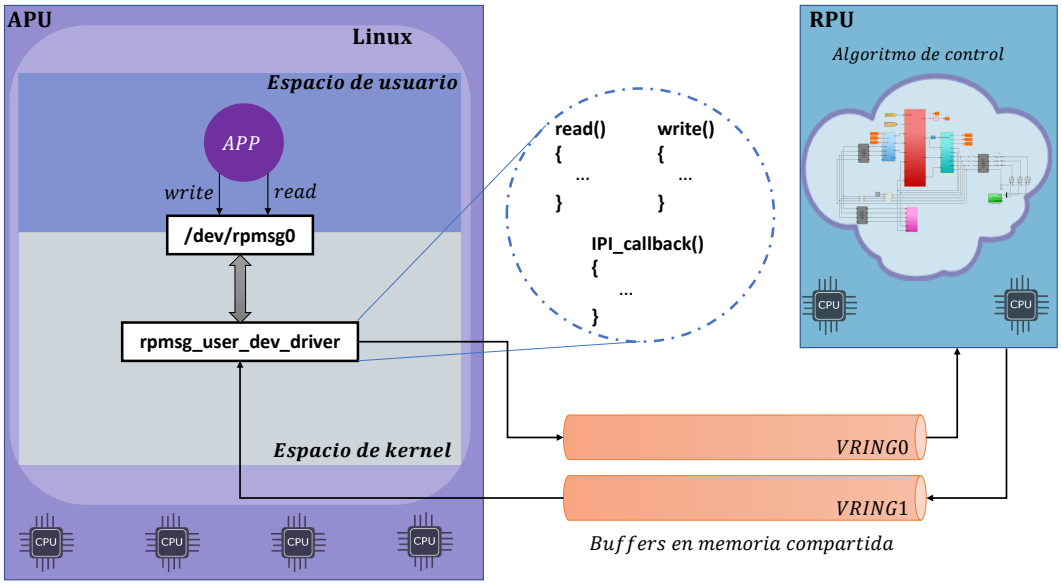


Figura 4.8: Flujo de comunicación *OpenAMP* en el *kernel*.

El contenido y el formato de la información intercambiada entre los algoritmos de control y los servicios en *Linux* se detalla en apartados posteriores.

4.4 OpenAMP en espacio de usuario

La implementación de la comunicación *OpenAMP* en espacio de usuario es radicalmente distinta a la descrita en 4.3. En este caso, se utiliza directamente la API proporcionada por la librería *OpenAMP* tanto en la aplicación desarrollada para *Linux* como en el código de control de los convertidores. La compilación del núcleo se realiza con el *device tree* que se muestra en el código 4.4. Se observa con claridad nuevamente la reserva de memoria para ubicar las secciones del código *baremetal* junto con los descriptores de *buffer* (*vring*) y la memoria compartida.

Listado 4.4: *Device Tree* para la implementación de *OpenAMP* en el espacio de usuario.

```
1  / {
2      reserved-memory {
3          #address-cells = <2>;
4          #size-cells = <2>;
5          ranges;
6          /* Reserved memory for both firmware and shared memory */
7          rproc_0_reserved: rproc@3ed000000 {
8              no-map;
9              reg = <0x0 0x3ed00000 0x0 0x8000000>;
10         };
11     };
12
13     amba {
14
15         vring: vring@0 {
16             compatible = "vring_uio";
17             reg = <0x0 0x3ed40000 0x0 0x40000>;
18         };
19         shm0: shm@0 {
20             compatible = "shm_uio";
21             reg = <0x0 0x3ed80000 0x0 0x80000>;
22         };
23
24         ipi0: ipi@0 {
25             compatible = "ipi_uio";
26             reg = <0x0 0xff340000 0x0 0x1000>;
27             interrupt-parent = <&&gic>;
28             interrupts = <0 29 4>;
29         };
30
31     };
32 };
```

En contraposición a la implementación en el núcleo de *Linux*, no se emplea el componente *Remoteproc* para controlar el ciclo de vida de los procesadores de la RPU, además de declarar de forma estática la ubicación de los *buffers* que contendrán los mensajes intercambiados entre los distintos núcleos (líneas 19-22).

Dado que en esta propuesta no se dispone de un componente como *Remoteproc* capaz de habilitar/deshabilitar la ejecución de código en la RPU en tiempo de ejecución de *Linux*, se debe generar un nuevo fichero *BOOT.bin* para incluir en la partición primaria de la SD. La creación del nuevo binario se realiza a través del *bootgen* integrado en SDK y un fichero de descripción .bif (de forma similar al proceso desempeñado para la transformación del *bitstream* con formato .bit a .bin:

```
bootgen -image sd_boot.bif -arch zynqmp -o .\BOOT.bin
```

El contenido del fichero con extensión .bif se adjunta en el código 4.5. El único elemento adicional con respecto al utilizado en la implementación en el *kernel*, es el ejecutable ELF correspondiente al código *baremetal* de control cuya ubicación de destino es uno de los *cores* R5 de tiempo real.

Listado 4.5: Código fichero .bif para la generación de BOOT.bin.

```
1 //arch = zynqmp; split = false; format = BIN
2 the_ROM_image:
3 {
4   [bootloader, destination_cpu=a53-0] .\zynqmp_fsbl.elf
5   [pmufw_image] .\pmufw.elf
6   [destination_cpu = r5-0] .\baremetal.elf
7   [destination_cpu = a53-0, exception_level = el-3, trustzone] .\bl31.elf
8   [destination_cpu = a53-0, exception_level = el-2] .\u-boot.elf
9 }
```

En esta propuesta, nada más encender la placa de desarrollo del sistema se pueden observar mensajes en la UART correspondientes al código de control corriendo en el núcleo R-5, antes incluso de iniciar el proceso de arranque del sistema operativo en la APU. La comunicación entre ambos contextos se realiza mediante el empleo de las librerías *OpenAMP*. Estas a su vez emplean la librería *libmetal* para el acceso a los recursos propios del canal RPMsg mediante el *framework* UserSpace I/O (UIO) permitiendo mapear en memoria los descriptores (*vrings*) y los propios *buffers* en espacio de usuario.

Para disponer de estas librerías en las aplicaciones desarrolladas para ser ejecutadas en *Linux* es necesario personalizar el sistema de ficheros en *Petalinux*, seleccionando los siguientes campos en la personalización del sistema de ficheros raíz:

```
Filesystem Packages --->
        libs --->
        libmetal   --->
                [*] libmetal
        open-amp   --->
                [*] open-amp
```

En la figura 4.9 se muestra el flujo de comunicación seguido en la implementación en el espacio de usuario. El acceso y la atención a las interrupciones se realiza mediante UIO.

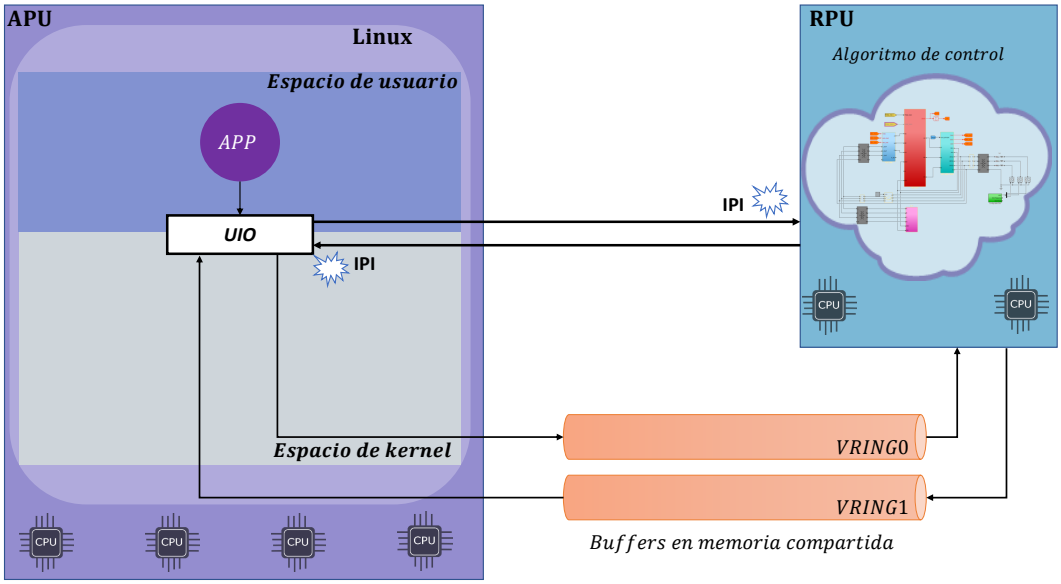


Figura 4.9: Flujo de comunicación *OpenAMP* en el espacio de usuario.

La principal motivación para la realización de esta alternativa es su flexibilidad y compatibilidad con

la virtualización de la plataforma, ya que el componente *Remoteproc* no es compatible con ella, por lo que si se quiere implementar la comunicación entre núcleos mediante las librerías *OpenAMP*, es la única solución posible a dia de hoy.

4.5 Protocolo de comunicación entre aplicaciones

En esta sección se incluye una descripción del formato de la información intercambiada entre los procesos de *Linux*, entre los que se incluye un servidor web y un servidor *ModBus*, con el código *baremetal* que implementa los algoritmos de control para los convertidores electrónicos de potencia de una micro-red. Así mismo, se comentan los fundamentos de ambos servicios y la interrelación/coordinación entre ellos con dos alternativas: plataforma basada en una aplicación monolítica o mediante micro-servicios.

El objetivo principal de los servidores (web y *ModBus*) es facilitar la interacción del usuario con la operativa de funcionamiento de cada convertidor. Por supuesto, estos servicios deben adaptarse a la aplicación concreta que se quiera realizar con los elementos de la micro-red. El intercambio de información se realiza de forma similar a la idea planteada en la comunicación *OpenAMP*, se declaran dos estructuras o paquetes de información asimétricos, uno de transmisión y otro de recepción, tanto en las aplicaciones de la APU como en el código de la RPU.

La asimetría viene impuesta nuevamente en función de la aplicación concreta, sin embargo, generalmente se dispondrá de una estructura de datos más extensa en recepción, en el sentido RPU $\longrightarrow$ APU para disponer de la lectura de las señales de tensión y corriente muestreadas así como parámetros internos de control que permitan verificar el correcto funcionamiento del sistema. Por otra parte, la estructura de transmisión se utiliza para enviar referencias de ciertas señales y comandar diferentes acciones al convertidor.

La estructura de transmisión en el sentido APU $\longrightarrow$ RPU requiere mecanismos de sincronización ya que ambos servidores envían datos a través de una misma estructura de datos. El mecanismo de sincronización utilizado depende de la infraestructura *software* desarrollada entre los procesos, por lo que su explicación se detalla en su apartado correspondiente.

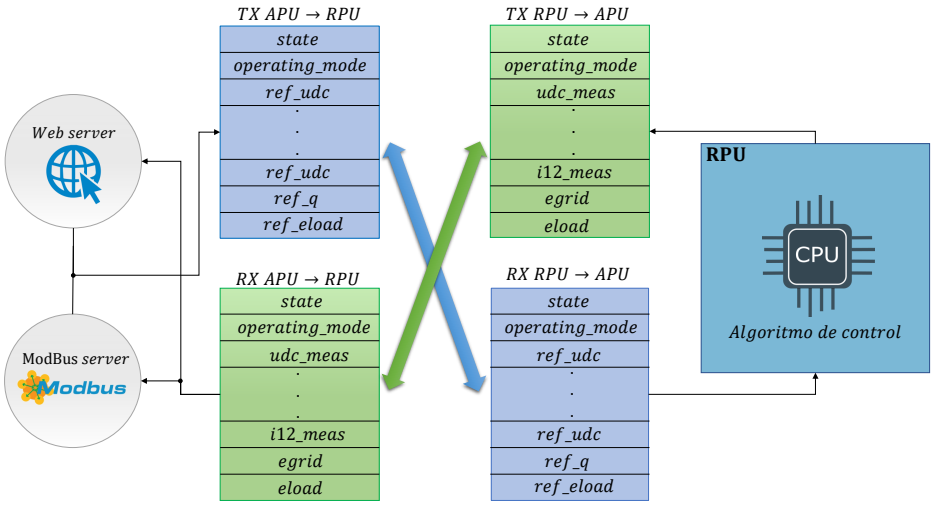
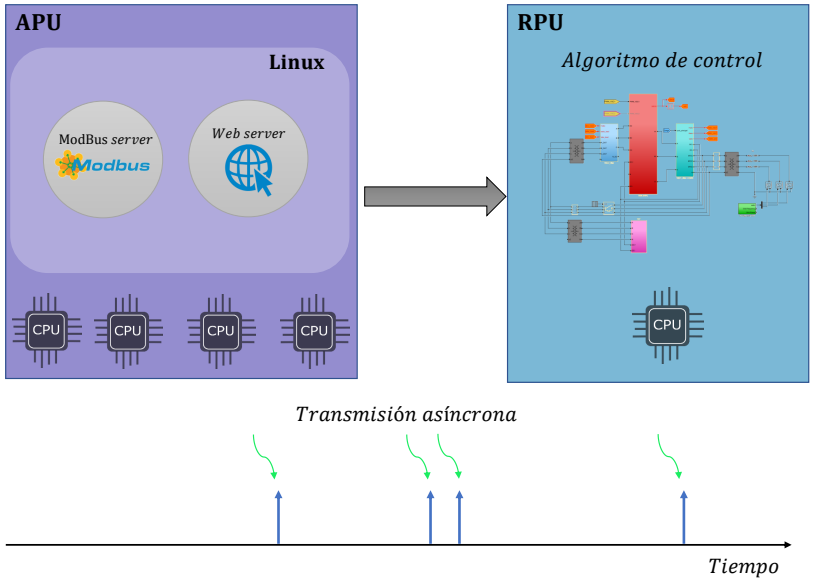


Figura 4.10: Esquema datos intercambiados entre los procesos de la APU y la aplicación de control de la RPU.

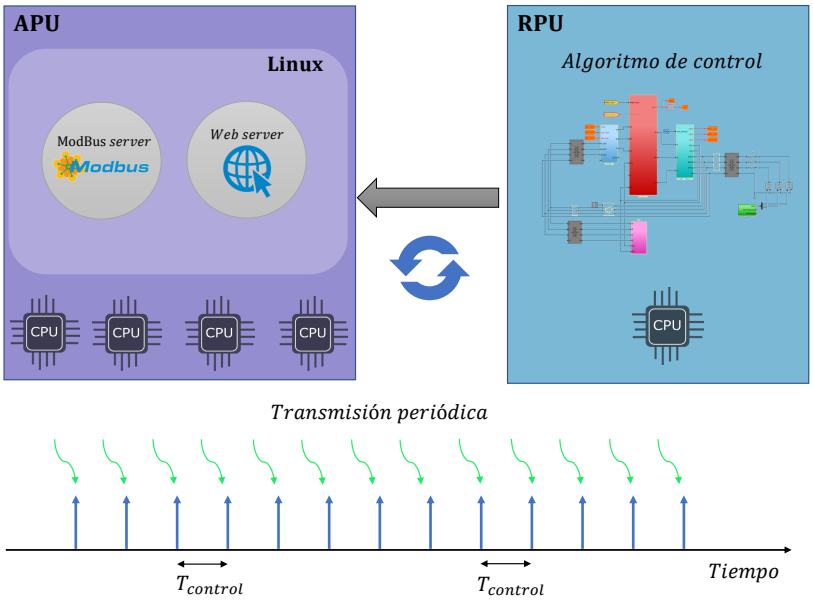
En la figura 4.10 se presenta un esquema simplificado de la información intercambiada en ambos

sentidos entre las aplicaciones en *Linux* y la desarrollada en *baremetal*. En concreto, la plataforma está basada en un envío de información continua desde la RPU hasta los servicios en la APU, es decir, cada periodo de control se actualizan los campos de la estructura de transmisión en el código *baremetal* y se reciben en las aplicaciones de *Linux* para su tratamiento/visualización en cualquier navegador web o en el cliente *Modbus*.

Por otro lado, el envío de información desde *Linux* tiene un comportamiento asíncrono, ya que únicamente se transmiten nuevos datos cuando el usuario interactúa con la plataforma para cambiar el estado/funcionamiento del convertidor o la modificación de alguna referencia de tensión, corriente, potencia ...etc. En la figura 4.11 se adjunta la diferencia entre el comportamiento de transmisión/recepción de datos en ambos sentidos: APU $\longrightarrow$ RPU y RPU $\longrightarrow$ APU.



(a)



(b)

Figura 4.11: Comparación comportamiento transmisión de información asíncrona (a) y síncrona (b).

Una de las partes más críticas del sistema es la velocidad máxima admisible en la recepción de la información desde el código *baremetal* hasta los servicios implementados en *Linux*, ya que se debe asegurar que todas las tareas se realicen en un tiempo inferior al periodo de control de actuación sobre los convertidores $T_{control}$ para que la plataforma cumpla los requisitos básicos de ejecutabilidad.

Este límite varía en función del tipo de implementación (*kernel* o espacio de usuario) de la comunicación *OpenAMP*, cuyas implicaciones se comentan en el capítulo 5.

4.5.1 Servidor web

El servidor web implementado en *Linux* sobre la APU sirve las páginas web que permiten al usuario visualizar el estado del convertidor y controlar el mismo mediante cambios de estado y envío de referencias. En función del método recibido, GET o POST, se modifica el contenido de la página en el navegador del usuario o se realizan ciertas acciones sobre el convertidor respectivamente. En la figura 4.12 se visualizan las acciones que realiza el servidor en función del método de la solicitud recibido.

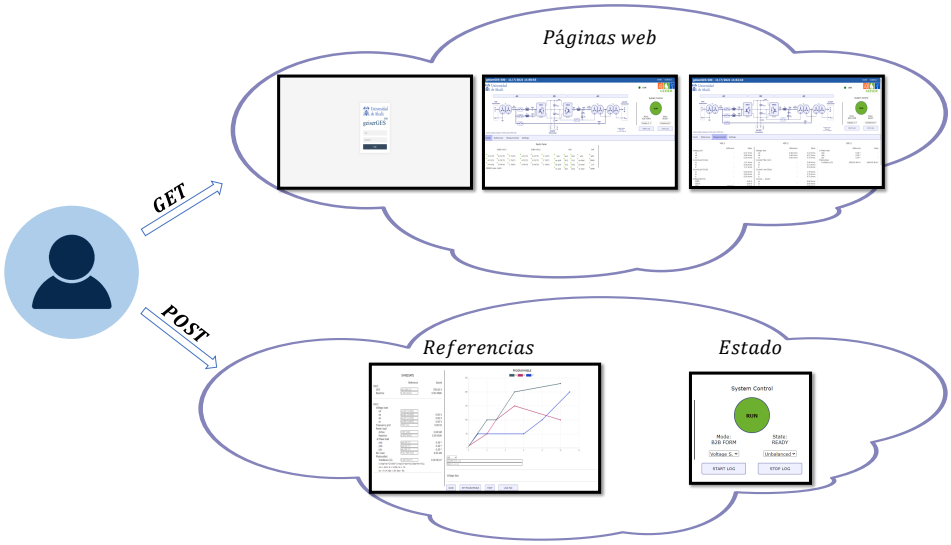


Figura 4.12: Esquema general del comportamiento del servidor web ante las solicitudes recibidas.

Cuando se recibe una solicitud de tipo GET, se brinda el recurso solicitado (el cual generalmente será el acceso a una nueva página web), mientras que una de tipo POST provoca siempre cambios en el servidor o acciones sobre el convertidor. En la tabla 4.2 se resumen las acciones que se realizan en función del tipo de solicitud

Método GET	**Método POST**
Solicitud de páginas web	Modificación de los estados del convertidor
Solicitud de imágenes y códigos *Javascript*	Modificación de referencias
Solicitud de actualización de gráficos	Posibilidad de creación de secuencias sobre cualquier referencia
Solicitud descarga de archivo de *logging*	Posibilidad de subir un archivo tipo .csv que realice múltiples secuencias de forma automática

Tabla 4.2: Comparación acciones en función de solicitud *GET* y *POST*.

La máquina de estados que define el funcionamiento del servidor web se muestra en la figura 4.13. Los diferentes estados por los que el servidor transcurre se describen a continuación:

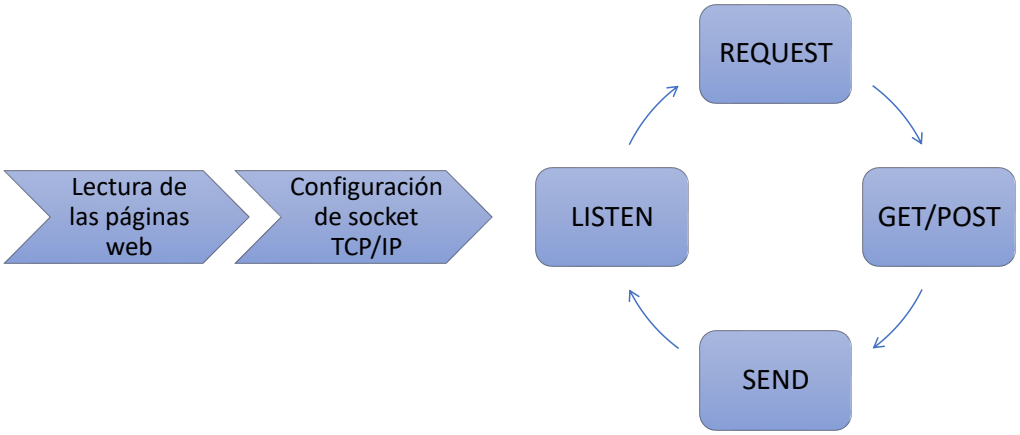


Figura 4.13: Máquina de estados del servicio web de la plataforma.

1. **Lectura de páginas web:** Las páginas que sirve el servicio web se encuentran almacenadas en el directorio *"web"* de la partición primaria *BOOT* de la SD. Una vez arranca el sistema, el contenido de la partición se monta en el sistema de ficheros de *Linux* y el servidor web las lee y almacena cada una de ellas en un array de carácteres con el tamaño adecuado.

2. **Configuración de socket TCP/IP:** En este punto el servidor web configura un *socket TCP* para aceptar solicitudes sobre la dirección IP de la placa de desarrollo y el puerto 8080. El máximo número de conexiones simultáneas se ha limitado a 5 usuarios.

3. **LISTEN:** se trata de un estado bloqueante a la espera de la recepción de una nueva solicitud por parte de los clientes web. Esto permite dormir el proceso del servidor cuando ningún usuario se encuentra conectado a la plataforma.

4. **REQUEST:** estado que procesa la solicitud HTTP recibida y determina si se trata de un método GET o POST.

5. **GET/POST:** en el caso de ser una solicitud GET se decodifica el recurso solicitado y se prepara para su transmisión al cliente. Si la solicitud es POST se utiliza la cabecera *"Content Length"* para determinar el tamaño del cuerpo del mensaje y se interpreta el mismo tanto para asignar ciertos valores en variables internas del servidor o realizar acciones sobre el convertidor mediante el envío de la estructura de transmisión al *core* R5 donde se encuentre ejecutándose la aplicación *baremetal*.

6. **SEND:** Se envían las respuestas HTTP junto con las páginas web asociadas en función de las solicitadas por el usuario. Finalmente, se cierra la conexión con el cliente.

La actualización dinámica de la información mostrada en el navegador del cliente se realiza mediante el empleo de una biblioteca de funciones escritas en *JavaScript* denominada *JQuery*, la cual permite integrar junto con las sentencias HTML de las páginas web, código que se ejecuta en la máquina del usuario.

Estas funciones se utilizan para modificar/actualizar elementos del panel de control de los convertidores en función del estado del mismo así como lanzar una solicitud periódica de forma continua que permita reemplazar la información mostrada en el navegador con su valor actual. En la implementación del servicio web se ha configurado una solicitud periódica de 500 milisegundos, al considerarla adecuada para el refresco de los datos.

4.5.1.1 Logging de datos

La comunicación entre el cliente y el servidor web es no conectiva (*connectionless*), es decir, el usuario abre una nueva conexión y envía una solicitud al servidor, que tras procesarla, envía su respuesta correspondiente para, finalmente, cerrar la conexión entre ambos extremos. Esta operativa es la que se realiza en la mayoría de los casos, exceptuando si se desea descargar un fichero de *logging* binario que incluya toda la información entre las estructuras intercambiadas por los contextos de procesamiento, APU y RPU, en tiempo real.

La descarga del fichero de *logging* requiere mantener la conexión abierta hasta que el cliente decida parar la transmisión de datos. En primer lugar, el servidor envía la cabecera de respuesta con los siguientes encabezados:

- **Código de estado:** 200 OK. Indica que la solicitud se ha procesado correctamente.
- ***Content Type:*** *octet-stream*. Se utiliza como valor por defecto para la transmisión de un archivo binario consistente en un flujo de *bytes*.
- ***Connection:*** *Keep-Alive*. Este encabezado se utiliza para indicar al cliente la forma de conexión de la comunicación establecida, para evitar que el cliente cierre la conexión mientras se está enviando el fichero de *logging*.

La velocidad de la descarga del fichero depende directamente del periodo de control utilizado para los algoritmos implementados en el código *baremetal* y el tamaño de las estructuras de transmisión (TX) y recepción (RX) intercambiadas entre las aplicaciones. Se debe tener en cuenta que se forma un paquete de información combinando las estructuras TX y RX para el envío al cliente cada periodo de control configurado. En la expresión 4.1 se muestra un ejemplo de la velocidad de descarga del fichero de *logging* con un tamaño de paquete de información de 600B y un periodo de interrupción de 100 μs:

$$Tasa_{binaria}(MB/s) = \frac{Tamaño\ paquete\ (B)}{Periodo\ de\ control\ (s)} = \frac{600\ B}{100e^{-6}\ s} = 6\ MB/s \tag{4.1}$$

En la infraestructura *software* del sistema desarrollada se debe asegurar en todo momento que se puede transmitir el paquete de información antes de que la aplicación *baremetal* transmita el siguiente estructura con la información de ese periodo de control, sino el sistema no cumpliría los requisitos mínimos de ejecutabilidad.

Una vez el usuario finaliza la descarga del fichero (mediante un botón para tal efecto sobre la página web), los datos se procesan y visualizan mediante un *script* en *Matlab* realizado en el grupo de investigación. La descripción de la etapa de post-procesado del fichero binario no se incluye en este documento.

El proceso de descarga del fichero de *logging* se muestra de forma gráfica en la figura 4.14, donde el usuario inicia la descarga del fichero mediante un botón en la página de control del sistema y finaliza de igual forma, pulsando otro botón para ese fin. La visualización de señales y parámetros internos se realiza mediante una etapa de post-procesado en Matlab.

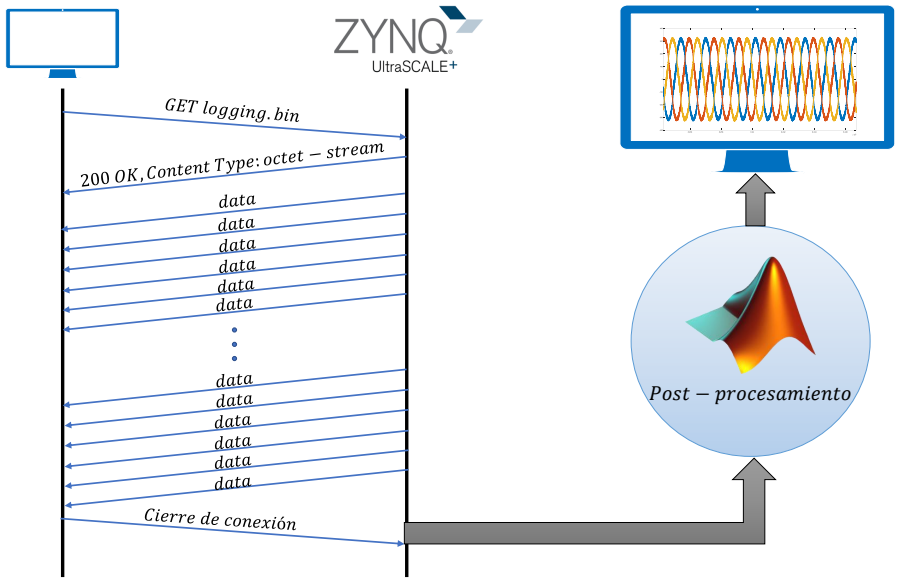


Figura 4.14: Proceso de descarga del fichero de *logging* desde el servidor web.

4.5.1.2 Modificación de referencias

El sistema implementado en este trabajo permite la modificación de referencias de los distintos algoritmos de control integrados en el código *baremetal*. Se puede diferenciar entre los cambios inmediatos o los basados en una secuencia.

Los primeros son los más sencillos, ya que únicamente consisten en asignar el valor introducido por el usuario (a través de una caja de texto en la página web) en la variable correspondiente de la estructura de transmisión para que el código ejecutándose en el procesador R5 de tiempo real lo aplique a la señal concreta y varíe el comportamiento del convertidor.

Por otro lado, se pueden definir secuencias para distintas referencias en formato array tipo *Matlab* o mediante la subida de un fichero .csv. En concreto, se debe enviar al servidor pares valor-tiempo que permitan construir la referencia a seguir. Desde el punto de la vista de la comunicación *OpenAMP* entre *cores*, *Linux* calcula dos valores para cada referencia:

- **delta_ref**: define el incremento/decremento que se debe aplicar a la referencia un número de veces igual al valor almacenado en "steps_ref".
- **steps_ref**: define el número de veces que debe ser aplicado el valor de "delta_ref" sobre la referencia.

Con los dos parámetros anteriores (es necesario reservar dos variables de este tipo para cada referencia que se permita realizar una secuencia sobre ella), se pueden comandar rampas y cambios bruscos sobre las señales con suma facilidad. La explicación de la obtención de estos valores a partir de los vectores introducidos por el usuario se realiza sobre la figura 4.15. En ella, se observan unos vectores de valor y tiempo de ejemplo, así como la forma de la referencia que se pretende realizar sobre el convertidor de potencia.

En cada tramo de la secuencia se calculan los valores de Δ y *steps* siguiendo las expresiones 4.2 y 4.3. Ambos valores se envían a través de la estructura de transmisión hacia el *baremetal*, cuya función es

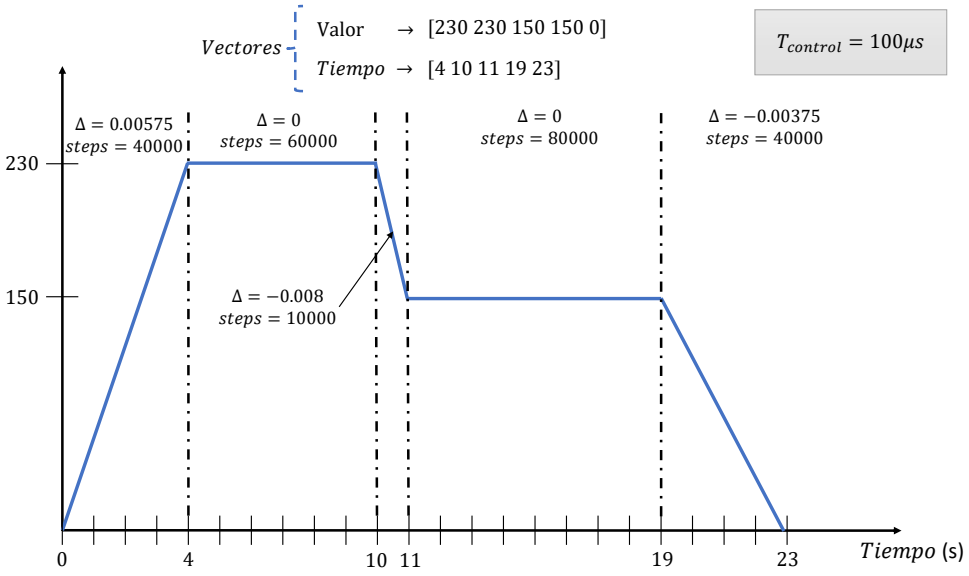


Figura 4.15: Ejemplo proceso de secuencia de una referencia.

sumar el valor de Δ en la referencia correspondiente en cada interrupción periódica de control, el número de veces que se indique en la variable *steps*.

$$\Delta = \frac{value_t - value_{t-1}}{tiempo_t - tiempo_{t-1}} \cdot T_{control} \tag{4.2}$$

$$steps = \frac{tiempo_t - tiempo_{t-1}}{T_{control}} \tag{4.3}$$

Cada vez que se suma una Δ, se decrementa el valor de *steps*, permitiendo conocer al contexto APU el estado de aplicación de la secuencia. Una vez la variable llega a 0, se vuelven a calcular las nuevas Δs y *steps* hasta completar la secuencia completa introducida por el usuario.

4.5.2 Servidor MODBUS

El sistema de control y comunicaciones desarrollado también incluye un servidor *ModBus* que incorpora una funcionalidad similar al servidor web. La razón de incluir este servicio es dotar a la plataforma de una mayor flexibilidad, pudiendo proveer distintos enlaces de comunicación con el exterior. La implementación de este servidor se ha realizado dentro del grupo de investigación *GEISER*, pudiendo encontrar en detalle una descripción del mismo en el trabajo [80].

En este documento únicamente se expone una vista general del funcionamiento del mismo y el mapeo que se realiza de los registros.

La máquina de estados que define el funcionamiento del servidor *ModBus* se muestra en la figura 4.16. Los diferentes estados por los que el servidor transcurre se describen a continuación:

1. ***mb_memory_mapping***: en este estado se realiza el mapeado en memoria de los registros *ModBus* (únicamente se utilizan *holding registers*) para lectura y escritura. En concreto, se reservan 256 registros para lectura/escritura y 65280 de solo lectura.

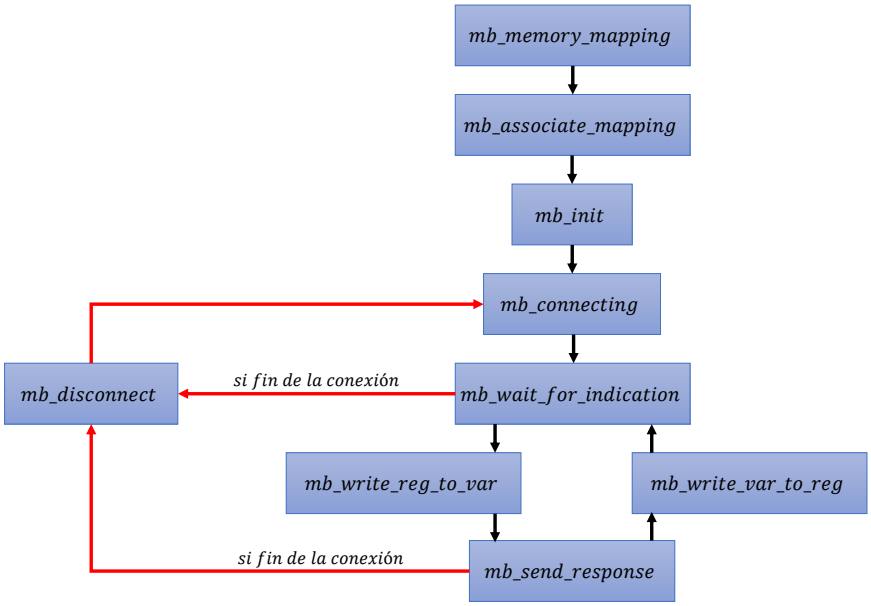


Figura 4.16: Máquina de estados del servicio *ModBus* de la plataforma.

2. ***mb_associate_mapping***: en este punto el servidor *ModBus* realiza la asociación de las variables de las estructuras TX y RX en *Linux* sobre un array de punteros con la dirección de cada una de ellas.

3. ***mb_init***: estado puente que realiza la inicialización del servidor.

4. ***mb_connecting***: se establece la creación y configuración de un *socket* TCP sobre el puerto 502. Se trata de un estado bloqueante hasta que un cliente establezca conexión con el servidor.

5. ***mb_wait_for_indication***: función bloqueante a la espera de una solicitud o *query* de lectura/escritura de registros.

6. ***mb_write_var_to_reg***: en este estado se atienden las solicitudes de lectura (código $0x3$) y escritura (código $0x10$) por parte del cliente. Primeramente, se actualizan los valores de los registros *ModBus* con los datos apuntados por el array de punteros inicializado en el proceso *"mb_associate_mapping"*. Si se trata de una solicitud de lectura, se pasa directamente al estado *"mb_send_response"*, si es de escritura, antes se llama a la función *mb_write_reg_to_var*.

7. ***mb_write_reg_to_var***: se encarga de actualizar las variables de las estructuras en memoria con la información introducida en los registros.

8. ***mb_send_response***: en este estado se envía la respuesta al cliente *ModBus*.

9. ***mb_disconnect***: estado que finaliza la conexión cliente-servidor antes de volver a la espera de una nueva conexión.

Las acciones que se pueden realizar con el servidor *ModBus* son equivalentes a las incluidas en el servicio web exceptuando la creación de secuencias y la descarga de un archivo de *logging*.

4.5.3 Plataforma basada en aplicación monolítica

En este TFM se proponen dos alternativas de infraestructura *software* entre los servicios implementados en *Linux*. Este apartado está centrado en la descripción de la plataforma basada en una aplicación

monolítica.

La arquitectura monolítica es considerada la alternativa más tradicional para la construcción de aplicaciones. Está basada en un conjunto único e indivisible, es decir, toda la funcionalidad se encuentra unificada en un único proceso. Los servicios web y Modbus, junto con la comunicación *OpenAMP* entre núcleos, se integran de forma conjunta mediante el empleo de hilos de ejecución en una misma aplicación. Este tipo de arquitectura se muestra en la figura 4.17, donde un único proceso está compuesto de los servicios de comunicación tanto externos (con un cliente externo a la placa de desarrollo) como internos (comunicación entre *cores*).

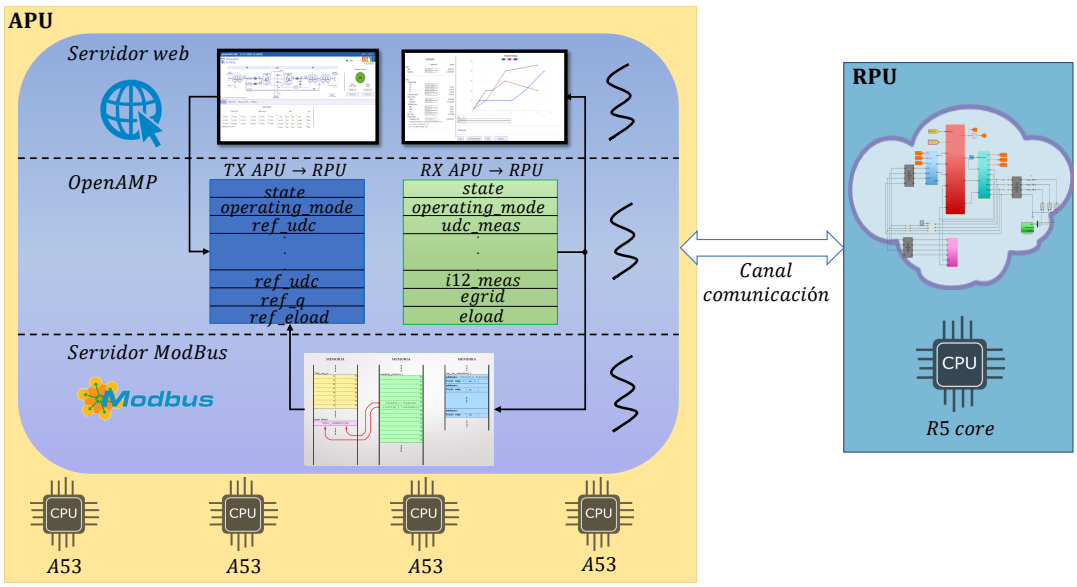


Figura 4.17: Plataforma basada en aplicación monolítica.

La funcionalidad de los hilos correspondientes a los servidores web y *ModBus* corresponde a la descrita en los apartados 4.5.1 y 4.5.2. El hilo de ejecución nombrado como *OpenAMP* es el responsable de recibir la información enviada por la aplicación *baremetal* a través del canal de comunicación RPMsg.

Tanto en la propuesta de implementación en el núcleo del S.O como en el espacio de usuario, consiste en un bucle infinito que se encuentra bloqueado a la espera de la recepción de nueva información. Los nuevos datos extraídos del canal, se almacenan en la estructura de recepción para que puedan ser utilizados por los servicios web y *ModBus* para actualizar la página web y los registros mapeados respectivamente.

Se debe tener en cuenta que este hilo se desbloquea de forma periódica, ya que como se expuso en la figura 4.11b, el código *baremetal* transmite de forma constante en cada interrupción de control.

Otra cuestión fundamental en esta arquitectura es el acceso a la estructura de transmisión por parte de los hilos web y *ModBus*. La problemática surge cuando varios usuarios, uno controlando la plataforma desde un navegador web y otro desde un cliente *ModBus* comandan un cambio sobre la misma variable, como pudiera ser un cambio de estado. La solución adoptada en la implementación es el empleo de semáforos sobre las sentencias de código que modifican parámetros de la estructura TX. En la situación en la que se quisiera modificar un mismo parámetro desde ambos hilos de ejecución en el mismo instante, únicamente se vería reflejado el cambio sobre la aplicación *baremetal* del último que accediera a la variable.

Las implicaciones así como las ventajas y desventajas de este tipo de infraestructura en términos de rendimiento y escalabilidad se comentan en el capítulo 5.

4.5.4 Plataforma basada en micro-servicios

La segunda propuesta de arquitectura *software* es la basada en micro-servicios, consistente en la división de una unidad monolítica en elementos independientes que interactúan y colaboran de forma conjunta. Cada elemento se suele denominar como el servicio de una aplicación. En la figura 4.18 se muestra la arquitectura de micro-servicios desarrollada en este trabajo.

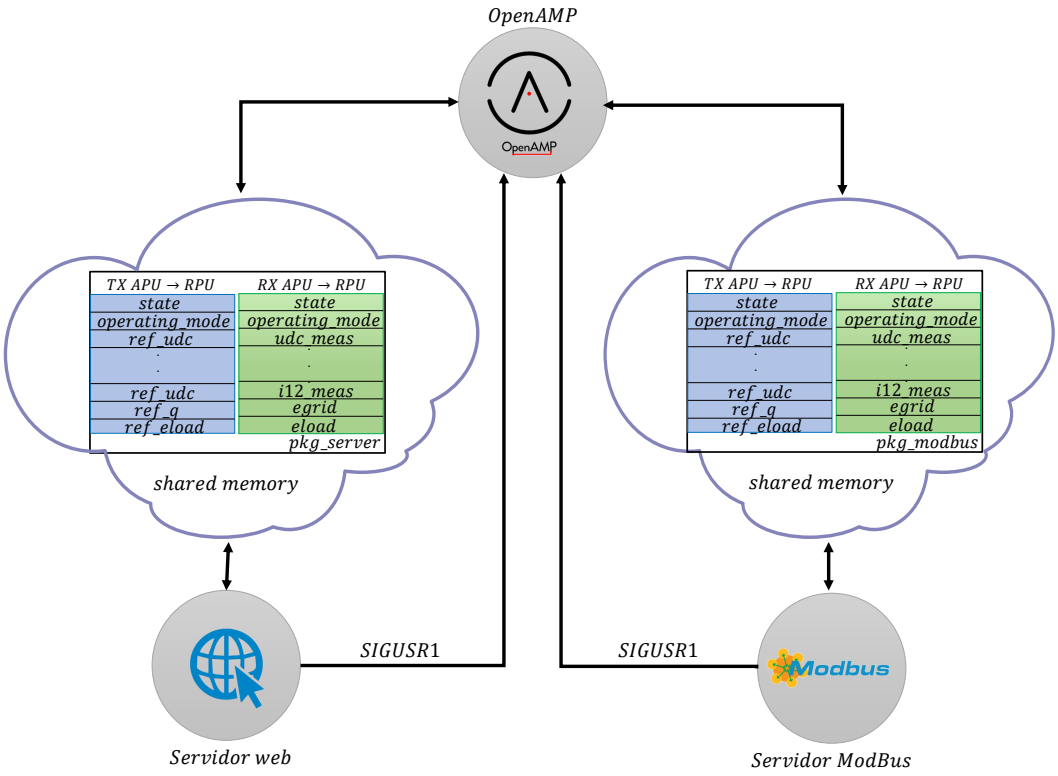


Figura 4.18: Plataforma basada en micro-servicios.

Cada uno de los servidores comparte con el servicio *OpenAMP* una zona de memoria compartida, donde se integran los paquetes de información formados por una estructura de transmisión y de recepción. En cada zona de memoria compartida, se asegura exclusión mutua, ya que el proceso *OpenAMP* escribe en las estructuras de recepción (color verde) y lee las de transmisión (color azul), de forma complementaria al acceso a los datos que realizan los servidores.

El servicio *OpenAMP* se encuentra esperando la recepción de información por parte del núcleo R5, de forma similar al hilo para tal efecto en la arquitectura monolítica. Una vez se obtienen los nuevos datos provenientes del *baremetal*, se copia en las estructuras RX de ambas zonas de memoria compartida para que los servidores dispongan de información actualizada. El diagrama de flujo correspondiente a la operativa expuesta con anterioridad se muestra en la figura 4.19.

La modificación del estado de los convertidores pertenecientes a la micro-red que se estén controlando, así como la comanda de referencias, tanto desde el servicio web como ModBus, siguen la misma operativa: una vez recibida la acción, se escribe en la variable correspondiente de la estructura de transmisión TX de la memoria compartida y se envía una señal SIGUSR1 al proceso *OpenAMP* junto con un identificador que permite reconocer que servicio la ha enviado. La recepción de esta señal, indica la necesidad de transmitir la estructura TX al *baremetal* para que actúe en consecuencia. En la figura 4.20 se adjunta un diagrama que resume el procedimiento seguido para modificar la funcionalidad de la aplicación *baremetal*.

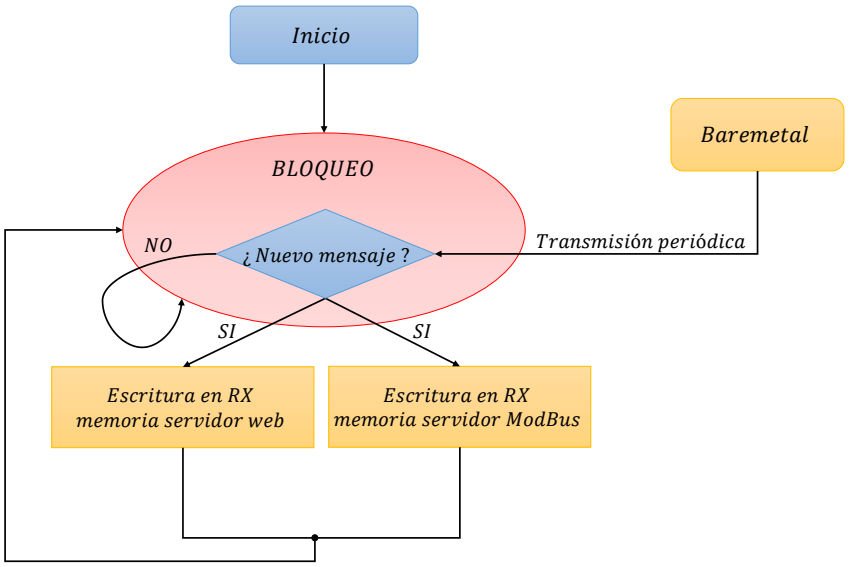


Figura 4.19: Diagrama de flujo correspondiente a la recepción de información desde la aplicación *baremetal* ejecutándose en el núcleo R5.

Con respecto a lo anterior, en este tipo de arquitecturas se debe mantener un sincronismo total entre los servicios para evitar problemas como el que se expone en el siguiente ejemplo:

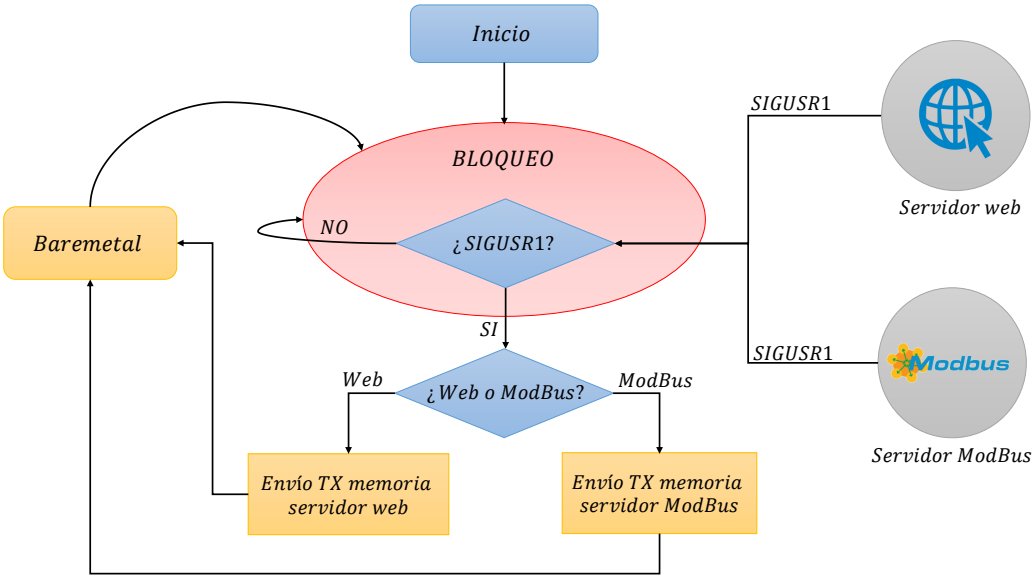


Figura 4.20: Diagrama de flujo correspondiente al envío de información por parte del usuario hacia el código *baremetal*.

Se plantea la situación en la que desde un navegador, un cliente desea modificar una referencia de potencia activa P, cuya variable asociada en la estructura de transmisión de la memoria compartida es *P_ref*. Por otro lado, otro usuario mediante un programa cliente *ModBus* desea modificar la referencia de potencia reactiva Q, cuya variable asociada es *Q_ref*. Además, las variables de las estructuras de transmisión de ambas zonas de memorias compartidas se encuentran inicializadas a 0. En este contexto, y a pesar de que la modificación de esas variables se realice en instantes de tiempo distintos, dado que el proceso *OpenAMP* envía a la aplicación *baremetal* la estructura TX de la memoria compartida del

servicio que haya enviado la señal SIGUSR1, el cambio en la segunda referencia provocará que la primera vuelva a 0, al pertenecer a espacios de memoria distintos. Lo anterior, se ilustra con mayor claridad en la figura 4.21, donde el envío de la referencia de potencia reactiva Q al *baremetal*, provoca que la referencia de potencia activa P se haga 0, cuando debería mantenerse con el valor introducido por el usuario a través del navegador.

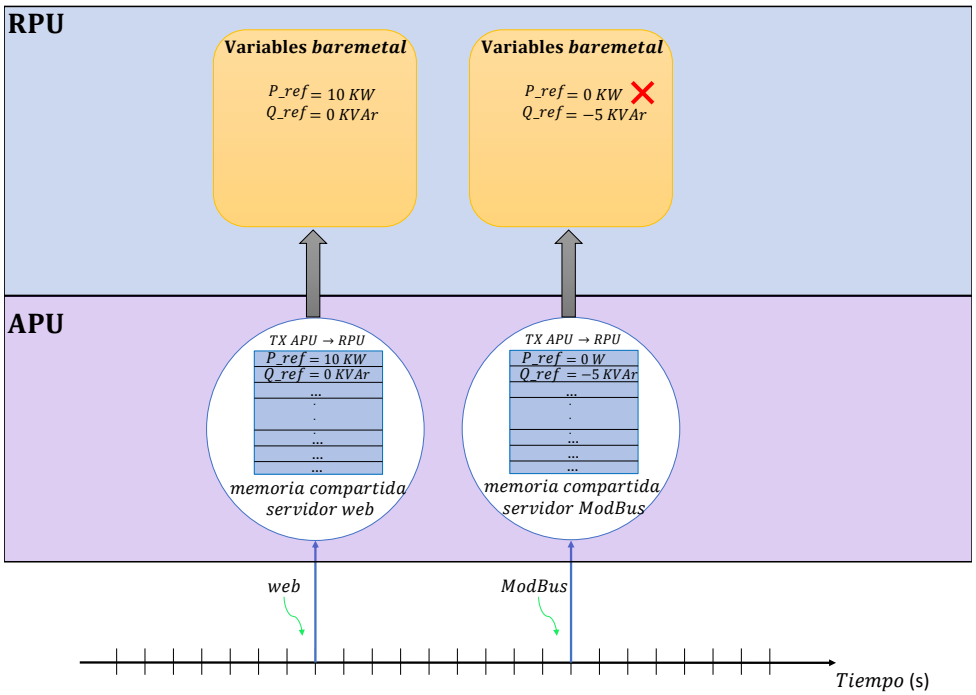


Figura 4.21: Ejemplo del problema de la falta de sincronismo en las estructuras de transmisión entre múltiples servicios.

La solución que se propone es sincronizar las estructuras TX en cada una de las regiones de memoria compartidas, con los campos que se comparten en la estructura RX, antes de modificar cualquier variable, con la finalidad de estar totalmente sincronizado con los cambios reflejados en la aplicación de control de los convertidores.

Esta arquitectura introduce una complejidad adicional para proporcionar algunas de las prestaciones comentadas en las secciones 4.5.1.1 y 4.5.1.2. Tanto el envío del fichero de *logging* como el principio de control de las secuencias, se apoyan en la temporización proporcionada por el bloqueo/desbloqueo del hilo *OpenAMP*. El desbloqueo del hilo se produce cada periodo de control utilizado en la aplicación *baremetal*.

Cuando el usuario introduce una secuencia de una determinada referencia, se activa un bit asociado a una variable de "modo" (incluida en la memoria compartida entre el servicio web y *OpenAMP*) y se envía una señal al proceso *OpenAMP* para iniciar y enviar los parámetros Δ y *steps* correspondientes.

Para el caso de la descarga del fichero de *logging*, la operativa es bastante dispar con respecto a la utilizada en la implementación monolítica. Primeramente, cuando el usuario desea registrar y descargar los datos en tiempo real, el servidor web activa el bit asociado al *logging* en la variable de "modo" y crea un hilo de ejecución asociado al envío del fichero. En ese instante, y tras enviar la señal *SIGUSR1*, se abre una FIFO (disponible en el sistema de ficheros de *Linux*) para lectura en el lado del servicio web y para escritura en el proceso *OpenAMP*, de forma que cada vez que se recibe nueva información por parte del *baremetal*, se escribe en la FIFO, mientras el hilo perteneciente al servidor web lee esos datos y los envía a través de la conexión TCP/IP abierta con el cliente. Finalmente, una vez el usuario pulsa la finalización

de la descarga del fichero binario, se desactiva el bit de *logging* y se cierran ambos extremos de la FIFO. En la figura 4.22 se muestra un esquema de los pasos que se siguen para la descarga del fichero binario de *logging* en este tipo de arquitectura.

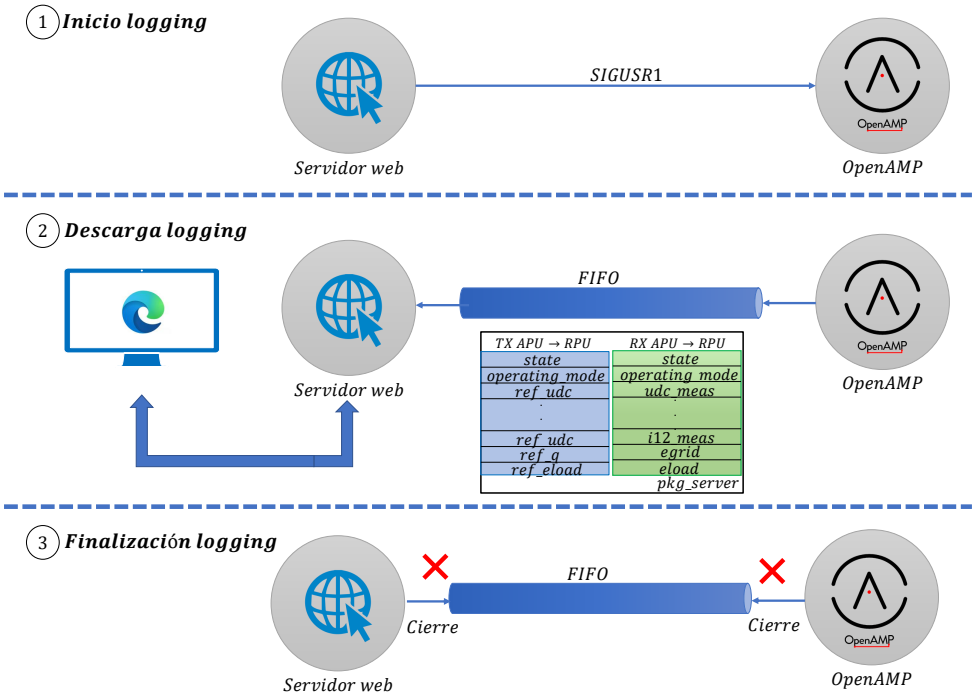


Figura 4.22: Flujograma de la descarga de un fichero de *logging* en la plataforma basada en microservicios.

4.6 Virtualización del sistema

En esta sección se expone la descripción de la metodología seguida para dotar a la plataforma de la virtualización, dotando al sistema completo de una mayor flexibilidad para ejecutar múltiples VM gestionadas por el hipervisor *Xen*.

Nuevamente, es preciso trabajar con el conjunto de herramientas *Petalinux* para la generación de forma sencilla de todos los elementos necesarios. En primer lugar, se debe habilitar el hipervisor *Xen* sobre el sistema de ficheros de la imagen de *Linux*, marcando la opción que se adjunta:

```
$ petalinux-config -c rootfs

  Petalinux Packages Groups --->
        packagegroup-petalinux-xen --->
                [*] packagegroup-petalinux-xen
```

Posteriormente, es necesario modificar a “INITRD” el tipo de sistema de ficheros:

```
$ petalinux-config

  Image Packaging Configuration  --->
    Root filesystem type (INITRAMFS)  --->
      (X) INITRD
```

Listado 4.6: *Device Tree* para la inclusión del hipervisor *Xen* sobre el sistema implementado.

```
/ {
    chosen {
        #address-cells = <2>;
        #size-cells = <1>;
        xen,xen-bootargs = "console=dtuart dtuart=serial0 dom0_mem=1G bootscrub=0 maxcpus=1
            timer_slop=0";
        xen,dom0-bootargs = "console=hvc0 earlycon=xen earlyprintk=xen maxcpus=1
            clk_ignore_unused";

        dom0 {
            compatible = "xen,linux-zimage", "xen,multiboot-module";
            reg = <0x0 0x80000 0x3100000>;
        };
    };
};

&smmu {
  status = "okay";
  mmu-masters = < &gem0 0x874
      &gem1 0x875
      &gem2 0x876
      &gem3 0x877
      &dwc3_0 0x860
      &dwc3_1 0x861
      &qspi 0x873
      &lpd_dma_chan1 0x868
      &lpd_dma_chan2 0x869
      &lpd_dma_chan3 0x86a
      &lpd_dma_chan4 0x86b
      &lpd_dma_chan5 0x86c
      &lpd_dma_chan6 0x86d
      &lpd_dma_chan7 0x86e
      &lpd_dma_chan8 0x86f
      &fpd_dma_chan1 0x14e8
      &fpd_dma_chan2 0x14e9
      &fpd_dma_chan3 0x14ea
      &fpd_dma_chan4 0x14eb
      &fpd_dma_chan5 0x14ec
      &fpd_dma_chan6 0x14ed
      &fpd_dma_chan7 0x14ee
      &fpd_dma_chan8 0x14ef
      &sdhci0 0x870
      &sdhci1 0x871
      &nand0 0x872>;
};

&uart1 {
  xen,passthrough = <0x1>;
};
```

Lo anterior, permite independizar el núcleo de *Linux* del sistema de ficheros del S.O, es decir, es necesario incluir en la partición primaria por un lado la imagen de *Linux* y por otro el *filesystem* del sistema.

Adicionalmente, se incluye un *device tree* denominado *"xen-overlay.dtsi"* (código 4.6) que describe la configuración e inicialización del hipervisor y el dominio virtual *Dom0*. En el código se incorporan una serie de argumentos de arranque tanto para el hipervisor *Xen* como para *Dom0*, los cuáles se comentan a continuación:

1. **Xen bootargs:**

   - ***console=duart***: especifica qué recurso *hardware* utilizará *Xen*.
   - ***duart=serial0***: indica el empleo de la UART0.

- ***dom0_mem=1G***: permite inicializar la cantidad de memoria RAM que se reserva para la utilización del dominio 0.
- ***bootscrub=0***: este parámetro permite liberar la memoria RAM libre durante el arranque. Se trata de una función de seguridad que evita que se filtren datos entre VM ante un fallo de *Xen*.
- ***maxcpus=1***: especifica el número máximo de CPUs que puede utilizar el dominio privilegiado.
- ***timer_slop=0***: ajusta el valor mínimo de los *timers* virtuales de *Xen*.

2. **Dom0 bootargs:**

   - ***console=hvc0***: se asigna la primera consola virtual del hipervisor al dominio privilegiado 0.
   - ***earlycon=xen***: consola utilizada en el arranque del dominio 0.
   - ***earlyprintk=xen***: se utiliza como método para visualizar información durante el arranque antes de que la consola por defecto (*hvc0*) se haya iniciado. Se suele utilizar para depuración.
   - ***clk_ignore_unused***: flag que permite ignorar los relojes de aquellos periféricos que se encuentren desactivados. Si no se incluye como argumento, en el arranque del sistema se deshabilitan todos los relojes correspondientes a los recursos *hardware* desactivados.

Así mismo en el nodo *chosen*, se incluye la propiedad *"compatible = xen,linuz-zimage, xen, multiboot-module"* junto con *reg* indicando la ubicación y el tamaño en memoria de la imagen de *Linux* que debe arrancar el dominio *Dom0*.

Por último, se incluye el nodo System Memory Management Unit (SMMU) junto con los nodos *hardware* específicos de la placa de desarrollo *ZCU102*. La SMMU permite gestionar el acceso a memoria traduciendo direcciones de memoria virtuales a las físicas reales.

Como se ha comentado en apartados anteriores, el despliegue de la plataforma junto con técnicas de virtualización sobre *Zynq UltraScale+* solo es posible con la alternativa de comunicación entre *cores* en el espacio de usuario, por ello, se debe incluir exactamente los mismos nodos que los incluidos en el *device tree* del código 4.4.

Los ficheros necesarios en la partición *BOOT* primaria de la SD son distintos con respecto a la alternativa sin virtualización. En esta ocasión se deben incluir los archivos que se describen a continuación:

1. ***BOOT.bin***: incluye los mismos elementos que se describieron en la tabla 4.1.
2. ***Image***: imagen de *Linux* que incluye el *kernel* del S.O.

   system.dtb: *device tree* compilado que incluye la descripción de todos los recursos *hardware*, así como los componentes *OpenAMP* de comunicación y de virtualización *Xen*.

   xen.ub: *bootloader* del hipervisor *Xen*.

   rootfs.cpio.gz.u-boot: *bootloader U-Boot* del sistema de ficheros del dominio *Dom0*.

En la figura 4.23 se visualizan los elementos integrados en la tarjeta SD para el despliegue de la plataforma de control y comunicaciones incluyendo el hipervisor *Xen*.

Dado que en este documento se incluye una gran variedad de alternativas y configuraciones de la plataforma desarrollada, el arranque del hipervisor se ha realizado de forma manual por comodidad, con la posibilidad de implementarlo de forma automática. El arranque manual del sistema consiste en indicar la ubicación en memoria de los distintos ficheros incluidos en la partición primaria, tal y como se adjunta en el siguiente comando:

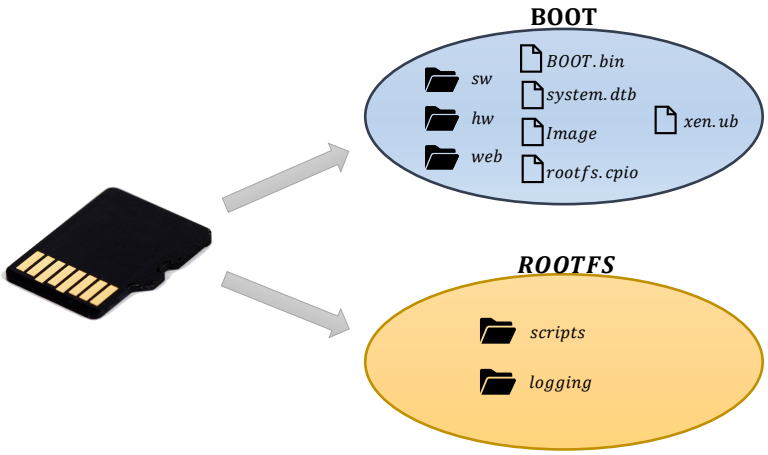


Figura 4.23: Particiones tarjeta SD para *Xen*.

```
ZynqMP> mmc dev $sdbootdev &&&& mmcinfo; load mmc $sdbootdev:$partid 1000000 system.dtb &&&&
load mmc $sdbootdev:$partid 0x80000 Image; fdt addr 1000000; load mmc $sdbootdev:$partid 1030000 xen.ub;
load mmc $sdbootdev:$partid 2000000 rootfs.cpio.gz.u-boot; bootm 1030000 2000000 1000000
```

Se puede observar como la secuencia de arranque se indica mediante el comando *"bootm"* con las direcciones de inicio de los ficheros incluidos en la partición.

Una vez iniciado el hipervisor y el dominio *Dom0*, se dispone de una imagen de *Linux* totalmente operativa con todos los privilegios para el acceso al *hardware* así como la posibilidad de gestionar y lanzar máquinas virtuales o *DomU*. Las acciones relativas a la gestión y despliegue de las VM se realiza mediante la herramienta *"xl"* incluida en *Xen*, cuyos comandos principales junto con su funcionalidad se detallan a continuación:

- ***xl create***: permite el despliegue de una nueva máquina virtual con un archivo de configuración que describa las características de la máquina. Un ejemplo de archivo de configuración se puede encontrar en el código 4.7. En él, se incluye el nombre de la máquina virtual, la memoria RAM asociada, la imagen del S.O, el número de vCPUs a utilizar e incluso la posibilidad de describir recursos *hardware* a través de un *device tree*.
- ***xl config-update***: posibilita actualizar los parámetros de un *DomU* concreto. No se realiza de forma inmediata sino que requiere el reinicio de la máquina virtual.
- ***xl destroy***: finaliza la ejecución del dominio sobre el que se aplica.
- ***xl list***: muestra un listado de los distintos dominios presentes en el sistema con información acerca de la memoria consumida, el tiempo de ejecución y las vCPUs asignadas.

Listado 4.7: Ejemplo archivo de configuración de un *DomU*.

```
1 name = "guest0"
2 kernel = "/boot/Image"
3 extra = "console=hvc0 rdinit=/sbin/init"
4 memory = 256
5 vcpus = 2
6 device_tree = "/boot/openamp-passthrough.dtb"
7 irqs = [ 61 ]
8 iomem = [ "0xff340,1", "0x3ed40,64", "0x3ed80,128" ]
9 vif = [ 'bridge=xenbr0' ]
```

En la figura 4.24 se muestra de forma visual las principales acciones que se llevan a cabo desde el dominio 0 principal con los comandos descritos con anterioridad.

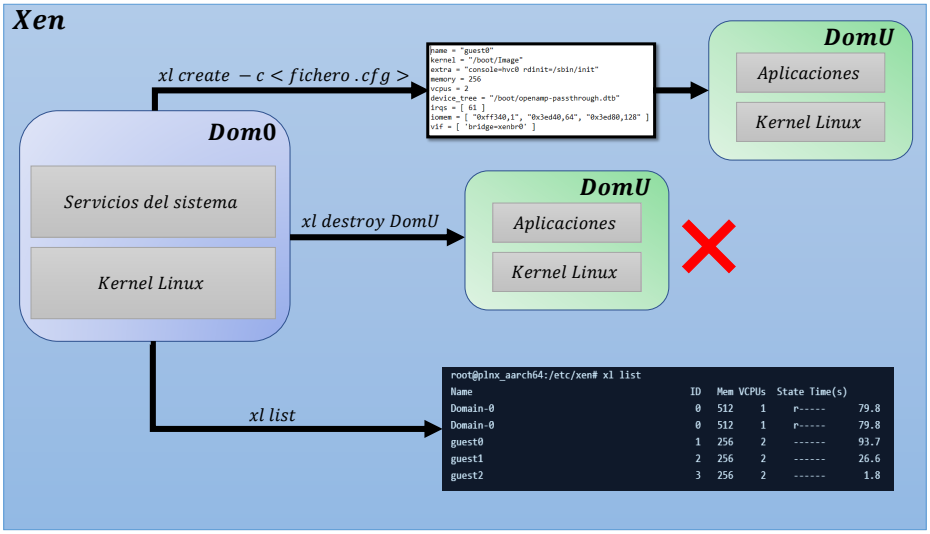


Figura 4.24: Gestión de máquinas virtuales a través del dominio privilegiado Dom0.

Las arquitecturas de la plataforma basadas tanto en una aplicación monolítica como en micro-servicios se integran sobre *Dom0* de forma equivalente a la descrita sin emplear el hipervisor *Xen*. Sin embargo, es necesario tener en consideración ciertos aspectos si se desea incorporar la infraestructura y la comunicación sobre un dominio no privilegiado (*DomU*).

Para ello, se debe modificar nuevamente el *device tree "system_user.dtsi"*, tal y como se muestra en el código 4.8, donde los componentes de la comunicación con el subsistema RPU se marcan como deshabilitados y con la propiedad *"xen, passthrough"*.

Posteriormente, a través de *Petalinux* se debe crear una imagen de *Linux* que incluya todos los componentes necesarios de *OpenAMP* (no es necesario incorporar ningún tipo de elemento adicional). Así mismo se debe generar un *device tree* denominado *passthrough* (código 4.9) que permita incluir los *vring*, los *buffers* y la IPI en el DomU.

Todos estos archivos se incluyen a su vez en la memoria SD para que se encuentren disponibles en el momento del arranque del dominio principal *Dom0* y se deben indicar en el archivo de configuración de la máquina virtual a desplegar (4.7).

Como se expuso en el apartado 3.9.1 del Estudio Teórico, el dominio *Dom0* es el único con privilegios para acceder a los recursos *hardware* por defecto, por lo que el envío/recepción de información de la plataforma a través del periférico de *Ethernet* siempre se realiza a través de él. Lo anterior, se observa con claridad en la figura 4.25, donde tanto la propuesta de la plataforma en *Dom0* como en *DomU* requiere el paso a través del primero.

Listado 4.8: Fichero *system_user.dtsi* para la implementación de la plataforma en DomU.

```
/{
    reserved-memory {
                #address-cells = <2>;
                #size-cells = <2>;
                ranges;
                rproc_0_reserved: rproc@3ed000000 {
                        no-map;
                        reg = <0x0 0x3ed00000 0x0 0x1000000>;
                };
        };
    amba {

                vring: vring@3ed40000 {
                        compatible = "vring_uio";
                        reg = <0x0 0x3ed40000 0x0 0x40000>;
                        status = "disabled";
                        xen,passthrough;
                };
                shm0: shm@3ed80000 {
                        compatible = "shm_uio";
                        status = "disabled";
                        xen,passthrough;
                        reg = <0x0 0x3ed80000 0x0 0x80000>;
                };
                ipi0: ipi@ff340000 {
                        compatible = "ipi_uio";
                        reg = <0x0 0xff340000 0x0 0x1000>;
                        interrupt-parent = <&&gic>;
                        interrupts = <0 29 4>;
                        status = "disabled";
                        xen,passthrough;
                };
        };

};
```

Listado 4.9: *Device tree* perteneciente a *DomU*.

```
/dts-v1/;

/{
        #address-cells = <0x2>;
        #size-cells = <0x2>;

        passthrough {
                compatible = "simple-bus";
                ranges;
                #address-cells = <0x2>;
                #size-cells = <0x2>;

                vring: vring@0 {
                        compatible = "vring_uio";
                        reg = <0x0 0x3ed40000 0x0 0x40000>;
                };
                shm0: shm@0 {
                        compatible = "shm_uio";
                        reg = <0x0 0x3ed80000 0x0 0x80000>;
                };
                ipi0: ipi@0 {
                        compatible = "ipi_uio";
                        reg = <0x0 0xff340000 0x0 0x1000>;
                        interrupt-parent = <0x1>;
                        interrupts = <0 29 4>;
                };
        };
};
```

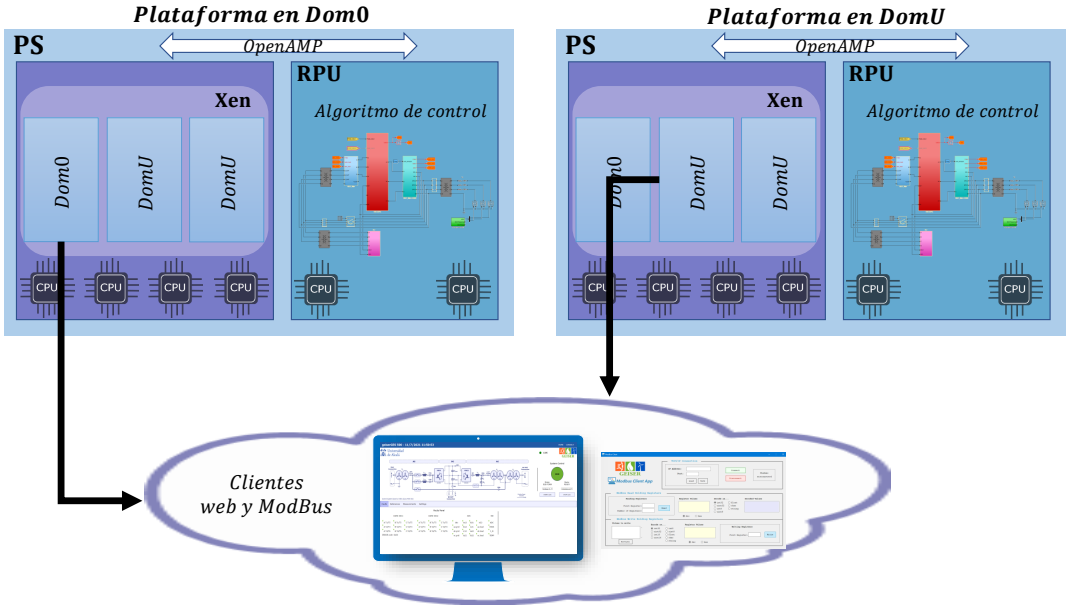


Figura 4.25: Comparativa comunicación con el exterior Dom0 y DomU.

Capítulo 5

Resultados

5.1 Introducción

En este capítulo se describen los resultados experimentales obtenidos mediante el sistema de control y comunicaciones implementado en este TFM.

Primeramente, se detalla el procedimiento seguido para la obtención de métricas en términos temporales que permiten evaluar las distintas alternativas propuestas en este documento, en especial, las implementaciones en espacio de *kernel* y usuario.

Seguidamente, la comparación entre las arquitecturas monolítica y micro-servicios se realiza desde un punto de vista conceptual, exponiendo las ventajas e inconvenientes en su empleo así como la exposición de resultados en términos de rendimiento de los servicios web y *ModBus*.

Por último, se describen las pruebas experimentales realizadas sobre un convertidor real, permitiendo evaluar la plataforma de control y comunicaciones desarrollada.

5.2 Métricas de calidad

En esta sección se definen las métricas de calidad empleadas para la evaluación de la plataforma implementada en este TFM.

En primer lugar, para la comparación de las implementaciones en espacio de usuario y de *kernel* se utiliza tanto una medida de temporización sobre el S.O *Linux* que proporcione información acerca de la cadencia de lectura de datos en las aplicaciones del sistema, como una medida de tiempos mediante el acceso a un *timer* implementado sobre la PL. La finalidad del *timer PL* es enviar en la estructura de transmisión desde el contexto RPU hasta la APU el valor de cuenta del temporizador justo antes de enviar el mensaje RPMsg, así como acceder al mismo una vez se lee dicha estructura en la aplicación de usuario. Lo anterior, proporciona una métrica de latencia desde que se envían los datos hasta que se reciben, permitiendo comparar el *driver* desarrollado por *Xilinx* (utilizado en la implementación del *kernel*) y el *driver* UIO (utilizado en la implementación sobre el espacio de usuario). El procedimiento seguido se detalla en el apartado 5.3.

De igual forma, se pretende analizar el rendimiento de ambas alternativas ante diferentes frecuencias de comunicación entre procesadores y tamaños de mensajes transmitidos, comparando el régimen de trabajo de ambas y la posible pérdida de paquetes por la sobre-escritura de información.

En segundo lugar, la comparación entre las infraestructuras monolítica y de micro-servicios se realiza principalmente mediante métricas cualitativas que describen las ventajas y desventajas de cada una de ellas. Así mismo, se incluyen representaciones temporales del consumo de la máquina de estados del servicio web.

Por último, se incluye una descripción del convertidor real sobre el que se ha verificado el correcto funcionamiento del sistema de control y comunicaciones, mostrando las diferentes pruebas y la funcionalidad de la plataforma.

5.3 Estrategia y metodología de experimentación

A continuación, se procede a comentar la estrategia de evaluación que se ha seguido. El entendimiento de las descripciones realizadas en este apartado es fundamental para valorar la calidad y la robustez del sistema.

La obtención de una métrica de calidad con respecto a la latencia en la recepción de la información en las aplicaciones implementadas sobre el espacio de usuario se realiza a través de la diferencia entre el valor de cuenta de un *timer* sobre la FPGA accedido en el momento en el que las aplicaciones han recibido los datos y el valor de cuenta del mismo justo antes de transmitir el paquete de información (expresión 5.1).

$$Latencia = \frac{cuenta_{recepción} - cuenta_{transmisión}}{frecuencia_{CLK}} \tag{5.1}$$

Para ello, es necesario tener acceso al *Timer* tanto desde la aplicación *baremetal* sobre el *core* R5 (el acceso al periférico es similar al efectuado con cualquier otro IP del diseño) como desde *Linux*. El acceso a las IPs desde el sistema operativo, integradas en el diseño *hardware* sobre la FPGA, requiere la modificación del núcleo de *Linux*, concretamente, el *device tree* del sistema. Se necesita superponer el nodo correspondiente al *AXI Timer* con la propiedad *"compatible=generic-uio"* así como asociar el módulo de interrupción *"uio_pdrv_genirq"* en los argumentos de arranque del sistema. En el código 5.1 se muestran los nodos y propiedades configuradas sobre el *device tree* para agregar la posibilidad de acceso al AXI Timer a través del *driver* UIO.

Desde el punto de vista de las aplicaciones, se deben utilizar las llamadas al sistema *open()*, *close()* y *mmap()* para abrir el descriptor de fichero asociado al UIO del *timer*, cerrarlo y mapear en espacio de usuario los registros asociados al AXI Timer respectivamente. En este caso, el único registro interesante es el asociado al valor de cuenta.

Además de lo anterior, se utilizan las funciones *clock_gettime()* declaradas en *<time.h>* para obtener marcas temporales que permitan analizar el comportamiento de los servicios de *Linux* ante la recepción constante de información.

Por otra parte, y dado que se pretende analizar la frecuencia máxima de transmisión de información y el tamaño máximo de paquete admisible por la plataforma desarrollada, se ha modificado el tamaño máximo por mensaje RPMsg, tanto en la implementación del canal RPMsg sobre el espacio de usuario como sobre el *kernel*, manteniendo un tamaño total máximo de la memoria compartida donde se alojan los *buffers* a un valor de 512KB (0x80000). En la tabla 5.1 se resumen todas las combinaciones de periodo de comunicación y tamaño del buffer utilizados para verificar el comportamiento de la plataforma.

Se debe tener en cuenta que para no superar el valor de 512 KB destinado a la memoria compartida entre los subsistemas APU y RPU a medida que aumenta el tamaño máximo de los mensajes deben

Listado 5.1: *Device Tree Overlay* para incorporar el acceso a los registros del AXI Timer.

```
/include/ "system-conf.dtsi"

/ {

  chosen {
    bootargs = "console=ttyPS0,115200 earlyprintk uio_pdrv_genirq.of_id=generic-uio";
  };

amba_pl: amba_pl@0 {

  axi_timer_0: timer@a0053000 {
    clock-frequency = <100000000>;
    clock-names = "s_axi_aclk";
    clocks = <&clk 71>;
    compatible =  "generic-uio";
    interrupts = <0 29 4>;
    reg = <0x0 0xa0053000 0x0 0x1000>;
    xlnx,count-width = <0x20>;
    xlnx,gen0-assert = <0x1>;
    xlnx,gen1-assert = <0x1>;
    xlnx,one-timer-only = <0x0>;
    xlnx,trig0-assert = <0x1>;
    xlnx,trig1-assert = <0x1>;
  };
};
```

disminuirse en la misma proporción el número total de *buffers*. La relación entre el nº de *buffers* y su tamaño se muestran en la tabla 5.2.

Periodo comunicación \ Tamaño del buffer	512 B	2048 B	8192 B	16384 B
1 ms	○	○	○	○
400 μs	○	○	○	○
200 μs	○	○	○	○
100 μs	○	○	○	○
50 μs	○	○	○	○

Tabla 5.1: Resumen de las pruebas realizadas en términos de periodo de comunicación y tamaño de buffer

Nº de *buffers*	**Tamaño máximo de *buffers***	Tamaño memoria compartida (Nº *buffers* x Tamaño *buffer*)
512	512 B	256 KB
256	2048 B	512 KB
64	8192 B	512 KB
32	16384 B	512 KB

Tabla 5.2: Relación entre el número y el tamaño de los *buffers* en la comunicación *OpenAMP*.

El análisis y estudio de los datos temporales obtenidos se realiza en Matlab mediante la descarga de un fichero de *logging* a través de la página web principal del sistema. En el fichero se incluye toda la información relevante obtenida o bien a través de las marcas temporales extraídas con la función *clock_gettime()* o bien mediante el acceso al AXI Timer a través del *driver* UIO. Finalmente, se debe mencionar que sobre la estructura de transmisión del código *baremetal* se ha incluido un campo que se incrementa cada vez que se envía un mensaje hacia el subsistema APU, de esta forma, se puede comprobar con facilidad si se pierde algún paquete en algunas de las combinaciones de frecuencia de comunicación y tamaño del mensaje comentadas con anterioridad. En la figura 5.1 se muestra un esquema que resume

la estrategia seguida para la obtención de resultados de índole temporal.

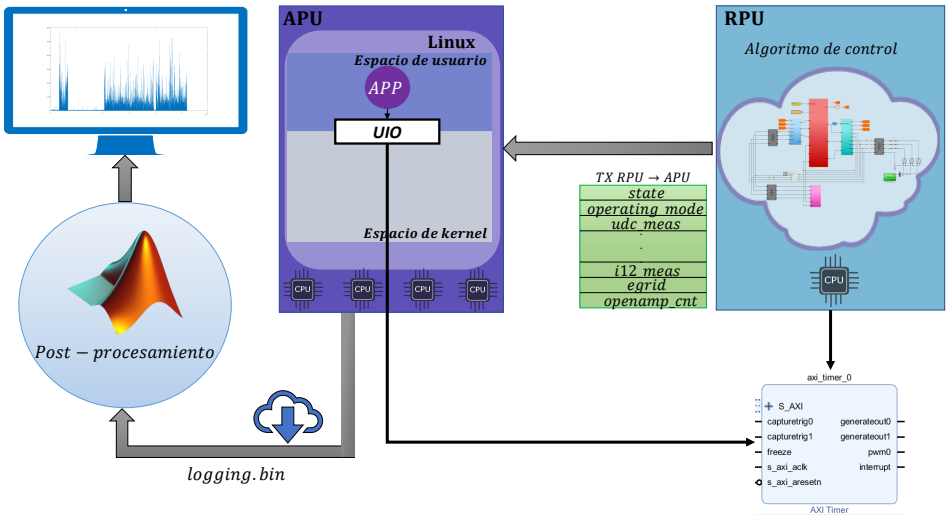


Figura 5.1: Esquema de la estrategia de experimentación de las alternativas propuestas en este trabajo.

5.4 Comparación de las implementaciones *OpenAMP*

La finalidad de este apartado es la presentación de los resultados temporales obtenidos tanto para la implementación sobre el espacio de usuario mediante las librerías *OpenAMP* y *libmetal* junto con el empleo del *device driver UIO* como la implementación en el *kernel* mediante *Remoteproc* y el *device driver "rpmsg_user_dev_driver"* desarrollado por *Xilinx*.

La denominación en espacio de usuario o *kernel* que realiza *Xilinx* está referida al protocolo *RPMsg*, no obstante, en ambas implementaciones se debe tener en cuenta que requiere el uso de un *device driver* en el espacio del *kernel*.

El límite inferior de periodo de comunicación utilizado para la extracción de los resultados, 50 μs, se ha escogido visualizando de forma experimental el tiempo total consumido por el código *baremetal* en cada interrupción periódica de control. En la figura 5.2 se adjunta el consumo temporal que supone las distintas partes de la aplicación ejecutada en el núcleo R5 (relativa al control del convertidor de potencia descrito en el apartado 5.6.1), donde se observa con claridad los escalones correspondientes al cambio de etapa en la máquina de estados implementada. Como puede apreciarse, en este caso concreto no podría implementarse una comunicación a 50 μs pues el sistema no sería ejecutable, sin embargo, se puede considerar un límite adecuado para la demostración de las posibilidades de la plataforma.

En las figuras 5.4a y 5.4b se muestra el tiempo desde que se bloquea el proceso o hilo (en función de la infraestructura utilizada) hasta que se dispone de la estructura enviada desde *baremetal* sobre los servicios en *Linux* utilizando como periodo de comunicación 200 μs. En ambos casos, se vislumbra con claridad como la proporción de escritura y lectura de los paquetes de información no es 1 a 1, es decir, la aplicación *baremetal* transmite una estructura y la aplicación en *Linux* es capaz de leer los datos antes de que se vuelva a escribir nueva información. Si fuera así, en las gráficas se vería un valor aproximadamente constante de 200 μs, sin embargo, se producen picos superiores al periodo de transmisión de información configurado. Lo anterior, es consecuencia directa de trabajar sobre un S.O gobernado por el *scheduler* o planificador que decide en qué momento y cuánto tiempo proporciona tiempo de CPU a los procesos en *Linux*.

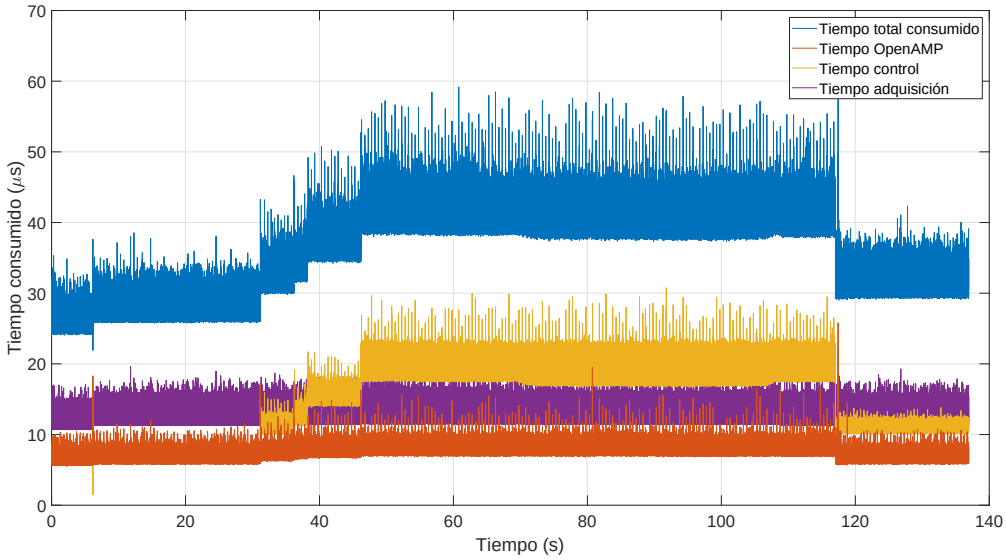


Figura 5.2: Tiempo consumido por el código *baremetal* en cada interrupción periódica de control.

En las situaciones en las que se produce un pico elevado en el tiempo de bloqueo, inmediatamente después se advierte una bajada brusca en las siguientes muestras, dado que en esas ocasiones se dispone de información para leer y no es necesario entrar en un estado de reposo esperando la escritura de nueva información.

Con el objetivo de comparar la latencia en ambas alternativas se han realizado varias pruebas modificando el periodo de comunicación con los valores que se expusieron en la tabla 5.1, sin tener en cuenta la posible pérdida de muestras y con un tamaño máximo de *buffer* de 512 B. El resultado obtenido se muestra sobre la figura 5.3.

A la vista de los resultados se observa con claridad como la alternativa basada en la implementación

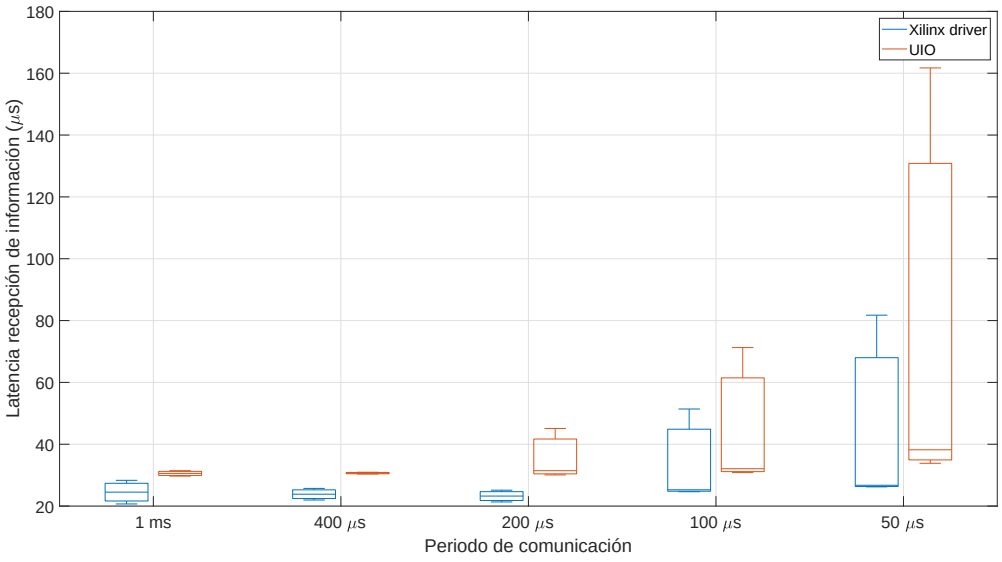
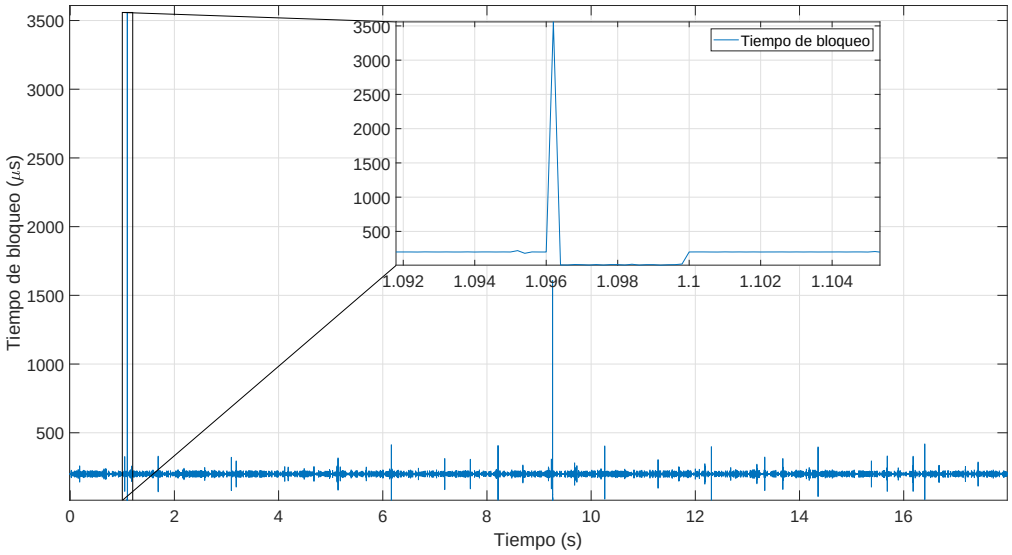
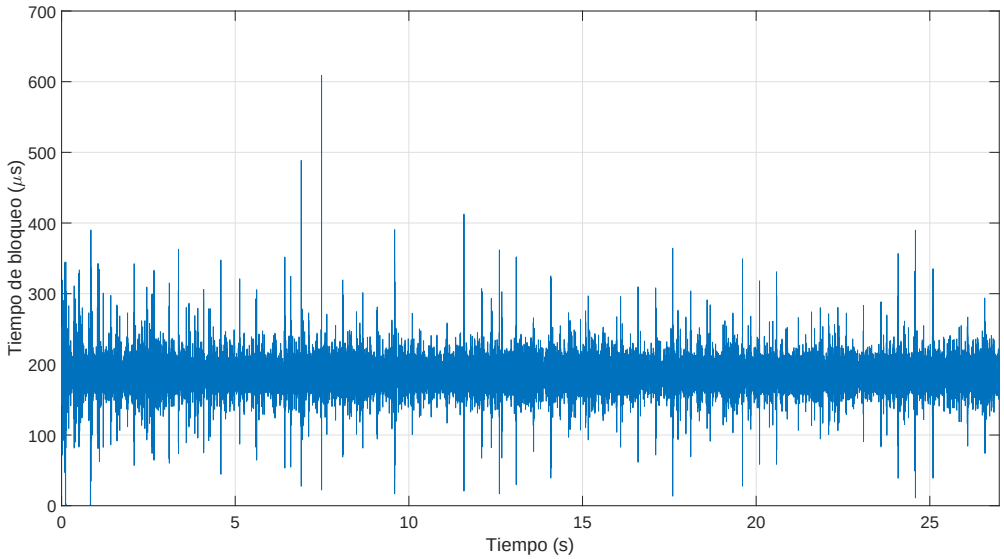


Figura 5.3: Comparación de la latencia entre la transmisión y recepción de la información entre ambas alternativas.



(a)



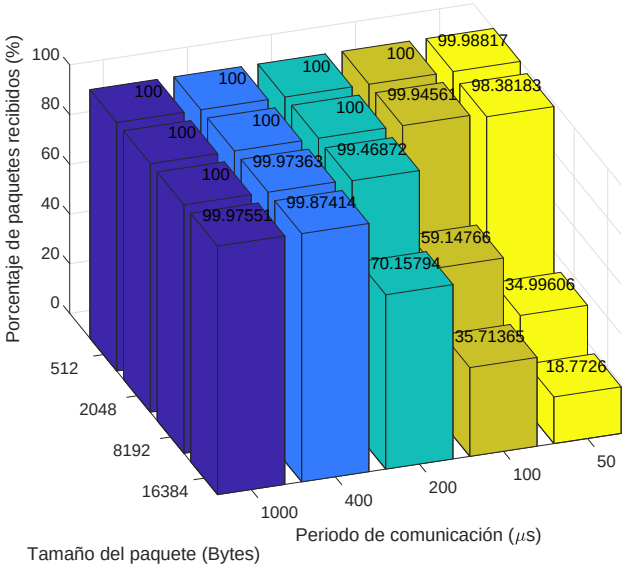
(b)

Figura 5.4: Comparación estado bloqueante en la aplicación de recepción de información: (a) RPMsg en espacio de usuario y (b) RPMsg en espacio de *kernel*.

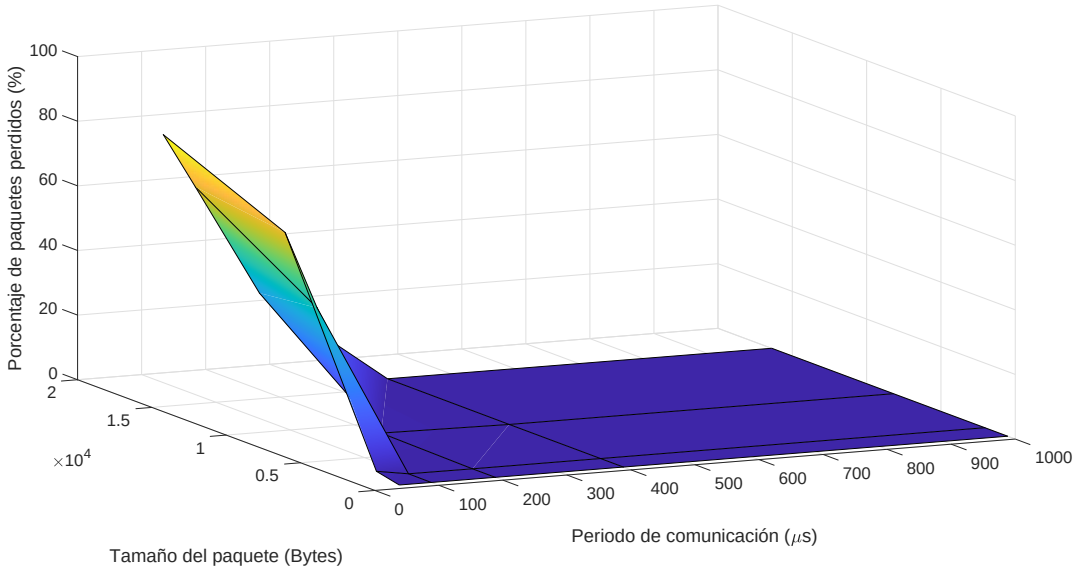
en el *kernel*, mediante el empleo del *device driver* proporcionado por *Xilinx* se consigue una latencia media inferior con respecto a la implementación en espacio de usuario con el uso del *driver* genérico UIO. En la gráfica 5.3 se indica con una barra horizontal la latencia media para cada uno de los casos mientras que el interior de cada una de ellas representa posibles valores que puede tomar en base a las pruebas experimentales realizadas. A medida que disminuye el periodo de comunicación entre los subsistemas APU y RPU la variabilidad en el tiempo es mayor, consecuencia nuevamente del planificador de *Linux* cuyo efecto se hace más apreciable al exigir una mayor velocidad en la transmisión de datos.

Para verificar el rendimiento y la consistencia de las estructuras recibidas en los servicios de *Linux* desde la RPU se ha calculado el porcentaje de paquetes recibidos y perdidos en función de la frecuencia de comunicación entre contextos y el tamaño del *buffer* RPMsg. Los resultados obtenidos se muestran en

la figura 5.5 para la alternativa RPMsg en el espacio de usuario y en la figura 5.6 para la propuesta en el *kernel*.



(a) Porcentaje de paquetes recibidos.

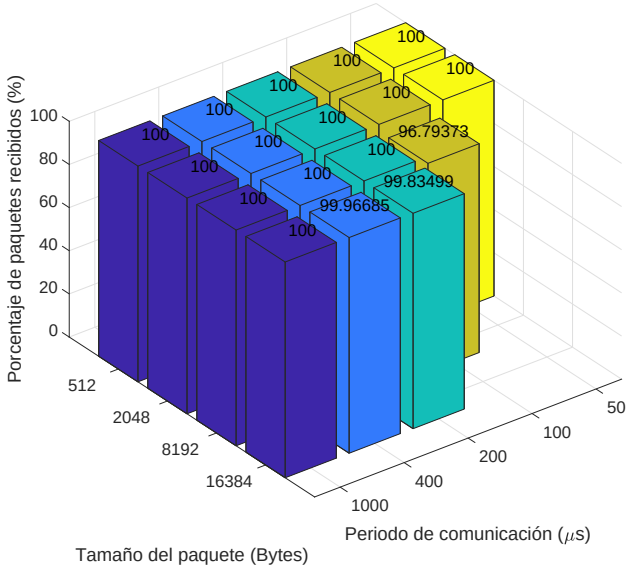


(b) Porcentaje de paquetes perdidos

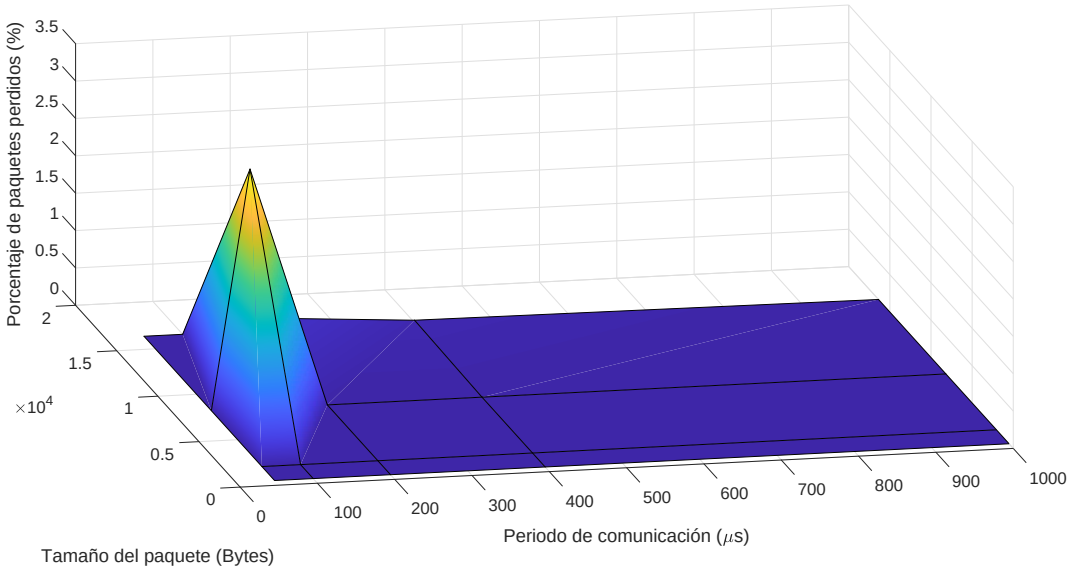
Figura 5.5: Rendimiento del sistema en la implementación de RPMsg sobre el espacio de usuario.

Con los resultados obtenidos, se puede verificar como el aumento en la frecuencia de la comunicación *OpenAMP* y la cantidad de información trasmitida afectan de forma considerable en el comportamiento del sistema implementado. En el caso de la alternativa RPMsg sobre el espacio de usuario se aprecia una reducción considerable en los paquetes recibidos de forma correcta. Tanto en 5.5a como en 5.5b se comprueba de forma clara el crecimiento en la tasa de pérdida de muestras o paquetes, provocado principalmente por la sobrescritura de los *buffers* compartidos entre ambos, pues la cadencia de la escritura es muy superior a la velocidad con la que las aplicaciones en *Linux* son capaces de leer los datos.

Por otro lado, la implementación RPMsg sobre el espacio del núcleo consigue unos resultados superiores (claramente reflejado sobre el plano de la figura 5.6b) donde se empieza a observar pérdida de



(a) Porcentaje de paquetes recibidos.



(b) Porcentaje de paquetes perdidos

Figura 5.6: Rendimiento del sistema en la implementación de RPMsg sobre el espacio del núcleo de *Linux*.

paquetes con un periodo de comunicación muy bajo y un tamaño de *buffers* elevado. Se debe destacar como las pruebas con un *buffer* de 8192 B y periodo de 50 μs así como con un *buffer* de 16384 B y periodo de 50 μs y 100 μs no se han podido evaluar debido a un fallo en la ejecución del *driver virtio_rpmsg_bus*, cuya causa no es motivo de estudio en este documento.

5.5 Comparación arquitecturas monolíticas y micro-servicios

En esta sección se realiza una descripción de las ventajas y desventajas que presentan la implementación de los servicios de *Linux* siguiendo una arquitectura monolítica o de micro-servicios teniendo en consideración que la funcionalidad es la misma. Además, se incluyen resultados cuantitativos con respecto al tiempo

consumido en la ejecución de la máquina de estados del servicio web así como el número de cambios de contexto y consumo de CPU de ambas arquitecturas.

Las principales ventajas que ofrece la implementación de una aplicación monolítica con respecto a una arquitectura basada en micro-servicios se adjuntan a continuación:

- **Mayor facilidad en la depuración y en el testeo.** Al trabajar únicamente con una única unidad indivisible es más fácil ejecutar la aplicación y comprobar posibles errores en el código ya que no es necesario preocuparse de la interacción entre las múltiples tareas que puedan existir, todas ellas se encuentran englobadas sobre el mismo proceso.
- **Simplicidad en el despliegue.** Una aplicación monolítica está contenida en un único archivo ejecutable, por lo que es relativamente sencillo lanzar su ejecución.
- **Comodidad en su desarrollo.** La implementación de aplicaciones monolíticas ha sido la alternativa por excelencia por su sencillez, por lo que cualquier desarrollador puede llegar a realizar cualquier tipo de aplicación de forma rápida y sencilla.
- **Mejor comportamiento ante cambios de contexto.** El multi-procesamiento en una arquitectura monolítica está basada en el uso de hilos, unidades ligeras de ejecución, mientras que una aplicación con micro-servicios se implementa con procesos, por lo que los cambios de contexto consumen un menor tiempo entre hilos pertenecientes a un mismo proceso.

Por otro lado, este tipo de arquitecturas también presentan desventajas que se deben tener en cuenta:

- **Dificultad en el entendimiento.** En la situación en la que la aplicación es muy grande, es decir, dispone de una gran funcionalidad (lo que supone un mayor número de líneas de código), es complicado conseguir un entendimiento completo cuando un desarrollador externo se enfrenta por primera vez al código.
- **Dificultad en los cambios.** A la hora de efectuar cambios sobre el código, pueden producirse situaciones difíciles por posibles acoplamientos entre partes de la aplicación, pudiendo producir errores imprevistos en la ejecución de la misma.
- **Escalabilidad.** Una plataforma monolítica no escala bien, ya que no puedes aumentar o disminuir el número de componentes de forma independiente.
- **Dependiente del lenguaje y la tecnología.** Una aplicación monolítica se escribe en su conjunto en un determinado lenguaje, si se desea aplicar un nuevo concepto de tecnología o de lenguaje se debe reescribir la aplicación al completo.

Una vez descrito las implicaciones en el empleo de una arquitectura monolítica, a continuación se presentan las ventajas de una aplicación basada en micro.servicios:

- **Mayor seguridad.** Una de las principales ventajas de los micro-servicios es el despliegue de componentes de la aplicación de forma totalmente independiente, sin que afecten unas en otras, es decir, un posible *bug* o error en alguno de ellos no tiene un impacto tan grande como en la monolítica.
- **Facilidad en su comprensión.** Al separar la funcionalidad completa de la aplicación en unidades más pequeñas es más fácil comprender el funcionamiento concreto de dicho componente.

- **Mayor escalabilidad.** Al independizar unos componentes de otros, se puede aumentar o disminuir el número de ellos sin ningún tipo de problema, teniendo en consideración las limitaciones del *hardware* físico subyacente donde se va a ejecutar la aplicación.
- **Independencia del lenguaje y la tecnología.** La aplicación basada en micro-servicios se encuentra dividida en componentes con distinta funcionalidad, por lo que es fácilmente adaptable, cada uno de ellos, a un lenguaje o tecnología distinta sin necesidad de reescribir el código de toda la aplicación, únicamente el componente concreto a adaptar.
- **Cambios de contexto más lentos.** La idea de micro-servicios con procesos independientes implica que la coordinación y ejecución de la aplicación supone cambios de contexto sobre las CPUs más pesados al ser procesos completos. Por supuesto, cada uno de dichos procesos pueden disponer de hilos ligeros de ejecución que supondrán un menor impacto en los cambios de contexto.

Para finalizar la descripción de las implicaciones de la arquitectura basada en micro-servicios se adjuntan las principales desventajas de esta alternativa:

- **Elevada complejidad de comunicación.** La aplicación basada en micro-servicios es totalmente distribuida, por lo que la comunicación y coordninación entre los distintos componentes es extremadamente compleja.
- **Dificultad en el despliegue.** En función de la aplicación, puede requerir un cierto orden a la hora de lanzar la ejecución de los servicios, sino se cumple no se garantizará un correcto funcionamiento.
- **Difícil depuración.** La depuración de cada unidad por separado es sencilla, el problema surge a la hora de testear todas ellos en conjunto.

Como puede comprobarse, las arquitecturas implementadas en este trabajo son totalmente opuestas, las ventajas de una son las desventajas de la otra, por lo que la decisión de emplear una alternativa u otra debe ser en base a los requerimientos que se planteen. En el contexto de las micro-redes, sería más adecuado la utilización de una arquitectura basada en micro-servicios que permita garantizar una separación e independencia entre los elementos del sistema eléctrico.

5.5.1 Comparación en el rendimiento en ambas alternativas

Primeramente, en la figura 5.7 se muestra la distribución del tiempo consumido por parte del servicio web tanto en la arquitectura monolítica como la basada en micro-servicios.

A la vista de la gráfica adjunta, el tiempo consumido por la aplicación monolítica es menor, sin embargo, no se aprecia una diferencia remarcable entre ellas, situándose por debajo de los 0.8 ms.

Posteriormente, para comparar en mayor detalle las dos alternativas se ha empleado el comando *"perf"* como herramienta de análisis de rendimiento sobre el S.O *Linux*. Se ha verificado el funcionamiento tanto en reposo, es decir, el servicio web en la página de control principal sin realizar ningún tipo de acción por parte del usuario, como realizando la descarga de un *logging* de datos con un intervalo de actualización de estadísticas de 2 segundos. Los resultados obtenidos se resumen en la tabla 5.3. En ella, se observa como en estado de reposo el rendimiento ofrecido por ambas arquitecturas es muy similar, con estadísticas muy parecidas, aunque se debe tener en consideración que los cambios de contexto en la alternativa monolítica son más rápidos al trabajar directamente con hilos ligeros de ejecución.

Las diferencias se empiezan a apreciar en el momento de realizar un *logging* de datos en tiempo real, los fallos de *branches* se disparan y el número de cambios de contexto también debido principalmente a

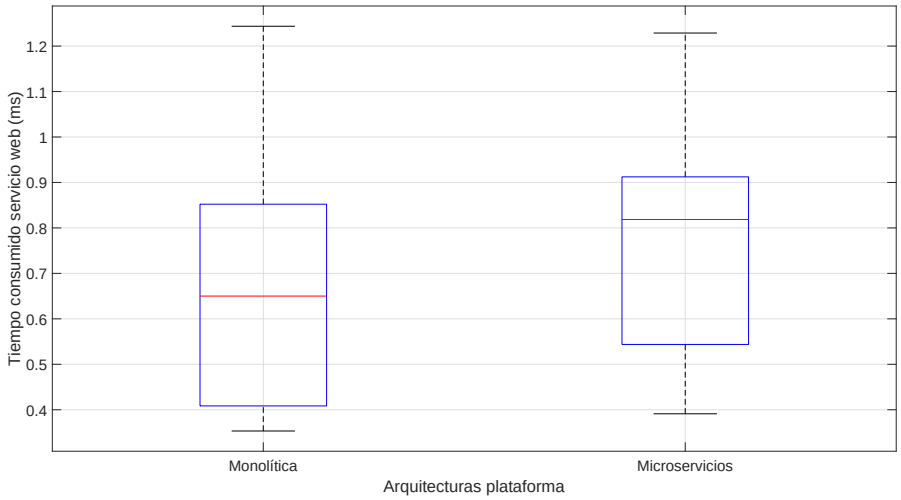


Figura 5.7: Comparativa del tiempo consumido en recorrer la máquina de estados del servicio web.

la necesaria comunicación entre los servicios *OpenAMP* y web (a través de una *FIFO*) para transmitir la información del *baremetal* al usuario.

Estadísticas	**Monolítica**		**Micro-servicios**	
	Estado de reposo	*Logging*	Estado de reposo	*Logging*
Cambios de contexto	0.126 M/s	0.126 M/s	0.126 M/s	0.166 M/s
Instrucciones/ciclo	0.31	0.64	0.32	1.81
branches	47.622 M/s	79.723 M/s	46.709 M/s	165.466 M/s
Fallos de *branches*	9.12 %	17.52 %	9.03 %	20.05 %

Tabla 5.3: Estadísticas obtenidas a través del comando *perf*.

A pesar de los términos de rendimiento anteriores, la elección de una u otra alternativa debe basarse en los requerimientos impuestos por la aplicación concreta. Por una parte, si se desea un desarrollo y despliegue rápido la mejor opción es la arquitectura monolítica, sin embargo, si se desea una implementación escalable y cuyos posibles errores en los módulos no afecten al sistema completo, la mejor opción es la basada en micro-servicios.

5.6 Pruebas experimentales sobre un convertidor real

En esta sección se incluye una descripción de la topología del convertidor así como su aplicación concreta para la emulación de red para ensayos de certificación. Primeramente, se realiza la exposición de las características del convertidor real. Seguidamente, se comentan los ensayos prácticos que se han llevado a cabo para verificar el correcto funcionamiento del sistema de comunicaciones y control sobre el SoC+FPGA. Finalmente, se muestran las gráficas de las pruebas realizadas, incluyendo las referencias enviadas por el usuario y las señales obtenidas a través del *logging* en tiempo real.

5.6.1 Descripción del convertidor

El convertidor sobre el que se ha implementado el sistema de control y comunicaciones detallado en este documento se trata de un convertidor Back-to-Back (B2B) trifásico de 3 niveles dimensionado para

trabajar hasta 700 kW de potencia, donde cada una de las ramas de los dos VSC siguen una topología ANPC. Este tipo de topología incorpora dos dispositivos de conmutación adicionales en paralelo a los diodos libres en un esquema DNPC (figura 3.7), sin embargo, debido a la modulación aplicada sobre ellos, se puede analizar su funcionamiento como un esquema DNPC tradicional ya que los nuevos IGBTs se dejan en abierto. La descripción y explicación detallada de la topología ANPC no está incluida en este documento. Un esquema general del convertidor electrónico se muestra en la figura 5.8, donde se incluyen las conexiones de alterna y continua a un equipo bajo test (DUT).

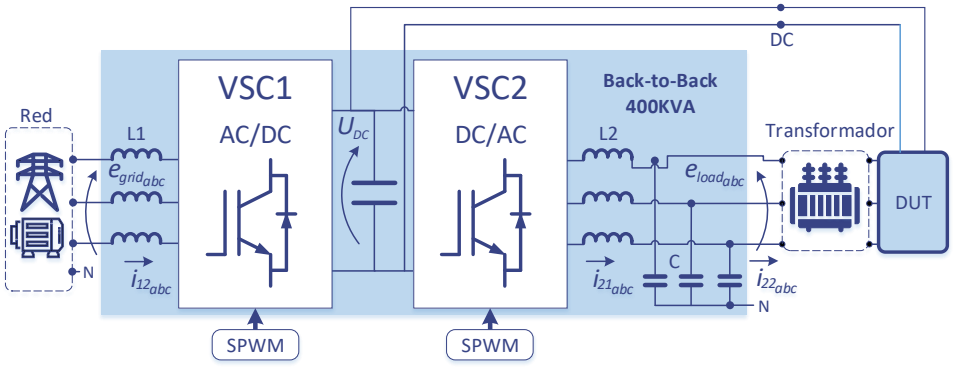


Figura 5.8: Esquema general del convertidor real. Fuente: extraída del *datasheet* creado en el grupo.

Las características principales del equipo se especifican en la tabla 5.4.

Entrada AC VSC1	Potencia máxima	50 kVA
	Tensión nominal	230 V_{fase} RMS
	Frecuencia nominal	50 Hz
	Factor de potencia	$-\pi/2 \ldots +\pi/2$ (ind./cap.)
	Límite tensión	320 V_{fase} RMS
	Límite corriente	180 A_{fase} RMS
Salida AC VSC2	Potencia máxima	400 kVA a 230 V_{fase} RMS 700 kVA a 400 V_{fase} RMS
	Rango de tensiones	0-400 V_{fase} RMS
	Rango de frecuencias	40-70 Hz
	Factor de potencia	$-\pi/2 \ldots +\pi/2$ (ind./cap.)
	Límite tensión	420 V_{fase} RMS
	Límite corriente	600 A_{fase} RMS
	THD	<1.5 % a potencia nominal
Salida/Entrada DC	Potencia máxima	700 kW
		700 kVA a 400 V_{fase} RMS
	Rango de tensiones	700-1500 V_{DC}
Pérdidas	IGBT	8 kW
	Filtro	5 kW
	Ventilación	13 kW

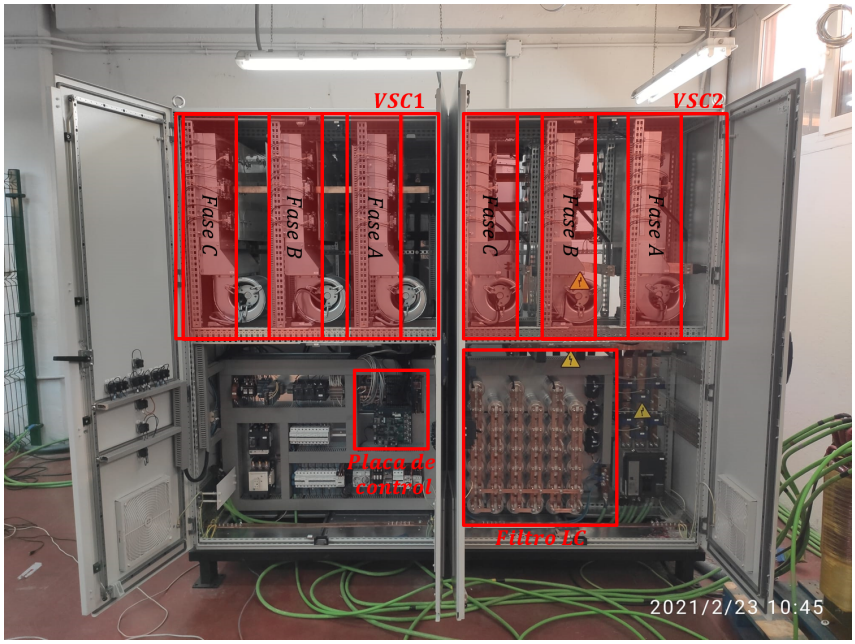
Tabla 5.4: Especificaciones del convertidor real.

En la figura 5.9 se muestran diferentes puntos de vista del equipo. En concreto, sobre la figura 5.9c se indican en color rojo los distintos elementos del convertidor, incluyendo las fases de cada VSC, el filtro

LC de salida y el sistema de control y comunicaciones.

(a)

(b)



(c)

Figura 5.9: Convertidor electrónico de potencia sobre el que se ha implementado el sistema de control y comunicaciones: (a) Vista lateral, (b) vista frontal y (c) equipo con las puertas abiertas

La gran cantidad de señales a medir en el convertidor B2B requiere incorporar al MPSoC+FPGA una placa interfaz, diseñada dentro del grupo GEISER. Como su propio nombre indica, actúa de interfaz entre el convertidor de potencia y el MPSoC. La placa contiene las siguientes características:

- 48 canales analógicos de entrada.
- 36 transmisores de fibra óptica.
- 18 receptores de fibra óptica.
- 4 relés de entrada.
- 4 relés de salida.

Las entradas analógicas se digitalizan a través de ADCs para disponer de las medidas de corriente y tensión como entrada a los algoritmos de control implementados en el código *baremetal*. Por otro lado,

los transmisores de fibra óptica permiten enviar las señales PWM hacia los *drivers* que actúan sobre los dispositivos de conmutación. De igual forma, los receptores de fibra se emplean para detectar posibles fallos sobre la actuación en los IGBTs. Por último, los relés de salida se utilizan para controlar la maniobra de los contactores principal y de carga del bus DC en la conexión entre la red eléctrica y el VSC1, mientras que los relés de entrada posibilitan verificar el estado de los contactores.

5.6.2 Descripción de pruebas experimentales

En este apartado se pretende describir la funcionalidad de cada uno de los VSC del convertidor B2B, así como las pruebas experimentales cuyos resultados se adjuntan en la sección 5.6.3.

En primer lugar, el VSC1 se utiliza como rectificador pasivo para realizar la pre-carga del bus DC a través de la conexión con la red eléctrica por medio de las líneas resistivas para tal propósito. Una vez finalizada la carga del bus, el VSC1 comienza a actuar con la funcionalidad de STATic COMpensator (STATCOM), aplicación ampliamente utilizada en sistemas de media-tensión que permite producir o consumir potencia reactiva en función de los requerimientos impuestos. Desde un punto de vista conceptual, el funcionamiento del VSC1 permite al usuario definir una serie de referencias de potencia reactiva (tanto positivas como negativas) así como mantener la tensión del bus DC constante en un valor de tensión indicado igualmente como referencia.

En segundo lugar, el VSC2 se emplea como una fuente de tensión AC configurable, cuya salida (tras el filtro LC) se conecta con el DUT a través de un transformador de relación 1:1. La finalidad del equipo es proporcionar un sistema de emulación de red para ensayos de certificación, por tanto, se permite la modificación de las tensiones, frecuencia y fase de las tres señales de salida para la realización de pruebas para huecos de todo tipo. El control del VSC2 se ha realizado con un controlador de tensión Full-Degree-of-Freedom (FDoF) [81].

La alternativa utilizada para el empleo del sistema de control y comunicaciones sobre el convertidor es aquella basada en una arquitectura monolítica con la implementación *OpenAMP* en el espacio del *kernel*.

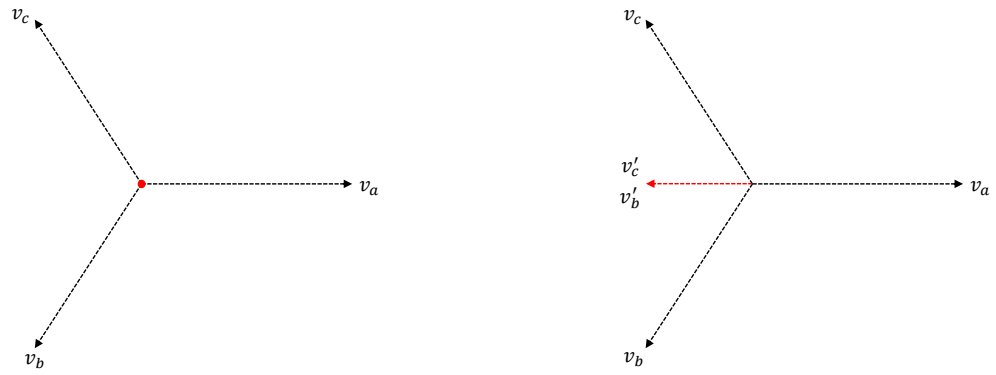
En este documento únicamente se incluyen las pruebas experimentales que se describen a continuación, como una pequeña demostración de toda la funcionalidad del equipo y la verificación del correcto funcionamiento de la plataforma desarrollada en este TFM:

1. Se emplea el VSC1 como rectificador activo, es decir, ante una consigna de tensión de continua para el bus DC, se sigue a dicha referencia, manteniendo su valor constante en el tiempo.

2. El VSC2 se utiliza para la emulación de las señales trifásicas de la red eléctrica, esto es, 230 V_{RMS} y 50 Hz de frecuencia fundamental.

3. La realización de huecos sobre las tensiones AC de salida del VSC2, tanto huecos trifásicos como bifásicos. En concreto, en la figura 5.10, se adjuntan las pruebas de huecos realizadas sobre el convertidor. La primera (figura 5.10a) consiste en un hueco trifásico del 0 %, todas las señales de salida se mantienen a 0 durante un cierto periodo de tiempo elegido por el usuario. La segunda (figura 5.10b) consiste en un hueco bifásico en el que es necesario modificar tanto la tensión como la fase de las señales v_b y v_c para compensar a la tensión v_a y mantener el sistema equilibrado.

La metodología y el procedimiento para el manejo de la plataforma se detalla en el manual de usuario (sección A), donde se incluye cómo introducir las secuencias de referencias de tensión DC, tensión AC en la salida del VSC2 y la modificación de las fases así como la monitorización del estado del convertidor.



(a) (b)

Figura 5.10: Pruebas experimentales de huecos sobre el convertidor (marcados en rojo): (a) hueco trifásico y (b) hueco bifásico.

5.6.3 Visualización de pruebas experimentales

La validación del correcto funcionamiento de la plataforma de control y comunicaciones desarrollada en este TFM se ilustra en las pruebas experimentales adjuntas en esta sección. Las distintas señales visualizadas se han extraído del fichero de *logging* descargado en tiempo real mientras se realizaban los ensayos sobre el equipo.

Primeramente, en la figura 5.11 se muestra el funcionamiento del VSC1 como rectificador activo. Se observa con claridad la primera etapa de pre-carga suave del banco de condensadores que forma el bus de continua. Una vez realizada la pre-carga, se realiza de forma automática una secuencia del valor de referencia de la tensión del bus, U_{DC}, desde el valor medido en ese instante hasta 700 V con una rampa de duración de 10 segundos.

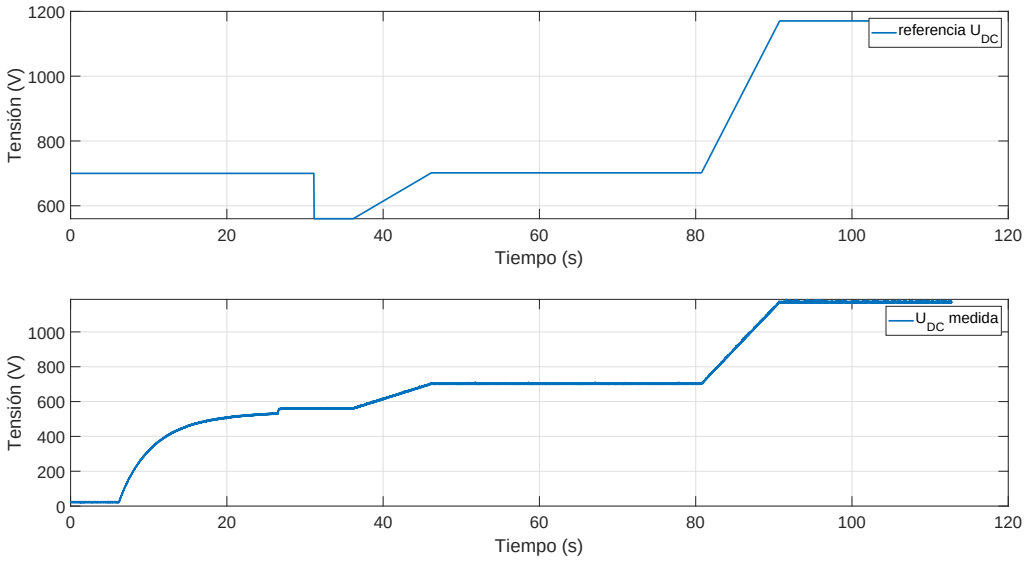


Figura 5.11: Funcionamiento rectificador activo del VSC1. En la parte superior se encuentra la referencia de tensión continua y en la inferior el valor de tensión medida en el bus DC.

Nuevamente, el usuario introduce una secuencia desde el valor actual hasta 1170 V en 10 segundos a partir de 80 segundos aproximadamente desde el comienzo de la prueba. A la vista del valor medido de tensión del bus de continua, se demuestra un seguimiento perfecto de la referencia por parte de los algoritmos de control implementados.

Por otro lado, en la figura 5.12 se adjunta las referencias introducidas en las señales de salida del VSC2 así como la medida alterna de dichas tensiones después del filtro LC. EN concreto, las tres referencias de tensión suben en forma de rampa de 0 V_{RMS} hasta 230 V_{RMS} con una rampa de duración de 5 segundos.

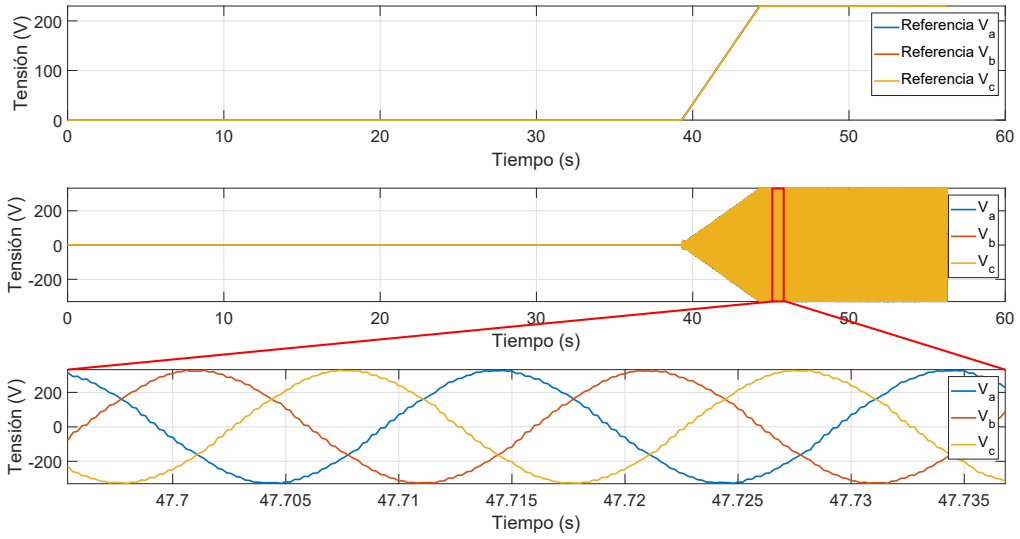


Figura 5.12: Funcionamiento fuente de tensión regulable VSC2.

Una vez más, el comportamiento del sistema de control y del convertidor es satisfactorio, consiguiendo un correcto seguimiento de la referencia de tensión de salida.

Los siguientes ensayos se centran en la emulación de posibles huecos sobre la red eléctrica para la certificación de equipos de potencia. La primera prueba, tal y como se expuso en 5.6.2, es la realización de un hueco trifásico, también denominado tipo A, consistente en una caída de tensión en las tres fases, v_a, v_b y v_c. En concreto, el hueco es del 0 %, donde todas las fases deben situarse en torno a una tensión nula.

En la figura 5.13 se muestra la referencia enviada desde la interfaz web hasta el *baremetal* y su resultado sobre las tensiones alternas de salida. En esta ocasión, son necesarios el empleo de escalones para producir un verdadero hueco sobre las señales. La configuración introducida consiste en una bajada y una subida de 1 ms y una duración de 500 milisegundos.

Sobre la gráfica inferior se puede observar como el control implementado consigue disminuir a 0, la tensión en la salida del VSC2 durante 500 ms.

El segundo ensayo efectuado consiste en un hueco bifásico, donde se debe modificar la ubicación de los vectores que representan a dos fases de las tensiones de salida, concretamente, la fase B y C. Para cumplir que el sistema se encuentre balanceado es necesario modificar tanto la amplitud de las tensiones como su ángulo. En la figura 5.14 se refleja el resultado de la prueba, donde en la gráfica superior se visualizan las referencias tanto de tensión como de diferencia de fase con dos escalas diferentes. En la escala izquierda se representan valores de tensión, mientras que en la parte derecha las unidades son grados de diferencia con respecto a la situación general de un sistema equilibrado. Esto último, supone

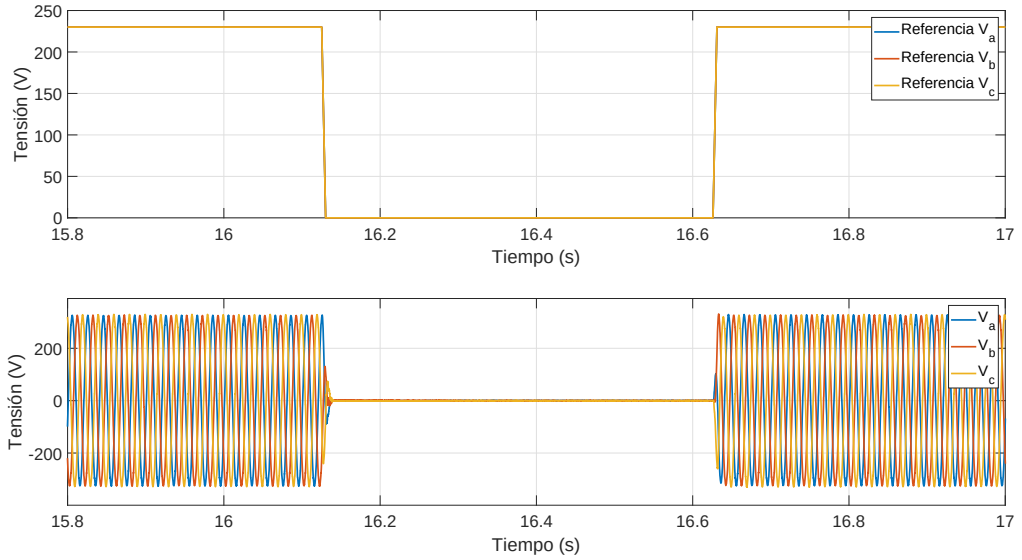


Figura 5.13: Demostración de la emulación de un hueco trifásico de 0º con una duración de 500 ms.

que el usuario debe introducir qué ángulo se debe aplicar a las tensiones del VSC2 con respecto al ángulo que tienen propiamente.

Dado que el objetivo es situar, tanto v_b como v_c en 180º, se deben aplicar unas referencias de -60º y 60º respectivamente, siguiendo el diagrama fasorial de 5.10b.

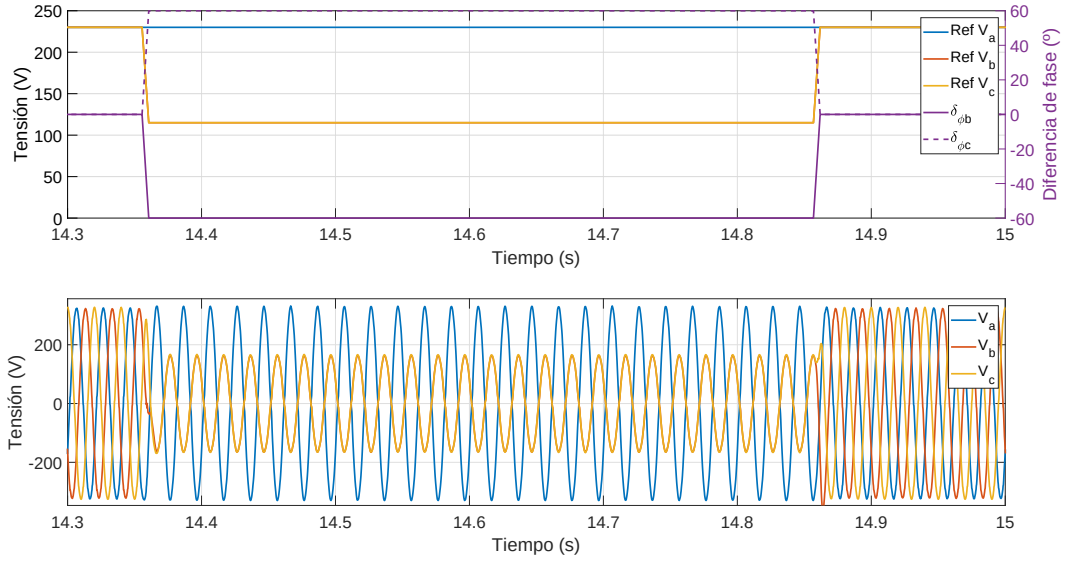


Figura 5.14: Demostración de la emulación de un hueco bifásico sobre las tensiones B y C, manteniendo el sistema balanceado.

En la parte inferior de la figura, se observa con claridad la correcta disposición de las señales medidas a la salida del filtro LC correspondiente al lado VSC2.

5.7 Conclusiones

Tras examinar los resultados previamente expuestos se advierte el buen comportamiento del sistema de control y comunicaciones implementado sobre un convertidor real B2B de hasta 700 kVA. De igual forma, se han realizado comparativas de rendimiento que permiten extraer las conclusiones más significativas en cuanto a qué tipo de implementación *OpenAMP* es superior y qué arquitectura de servicios sobre un S.O proporciona mejores prestaciones:

1. **Implementación *OpenAMP*:** tras el análisis de rendimiento realizado en ambas alternativas y a la vista de los resultados mostrados en cuanto a la pérdida de paquetes en función de la frecuencia de la comunicación entre distintos *cores*, se puede concluir que el empleo de un *device driver* sobre el *kernel*, encargado de controlar la mensajería RPMsg subyacente proporciona un rendimiento superior, consiguiendo evitar la pérdida de muestras a un periodo de comunicación de hasta 50 μs. Sin embargo, no se debe olvidar que la propuesta basada en el espacio de usuario con el uso de UIO proporciona la compatibilidad necesaria para incorporar al sistema las ventajas de la virtualización, extendiendo el control a un número mayor de convertidores pertenecientes a una micro-red. En función de las necesidades de cada micro-red será necesario trabajar con una u otra alternativa.

2. **Arquitecturas de servicios:** en las pruebas realizadas sobre el convertidor real se ha utilizado una arquitectura monolítica por su mayor simplicidad y rápido desarrollo para cumplir requisitos temporales del proyecto al que pertenece el diseño del equipo. Sin embargo, dado que la diferencia en rendimiento entre la arquitectura monolítica y la basada en micro-servicios no es muy significativa, una buena opción para continuar con el desarrollo del sistema es la de micro-servicios, al incorporar la independencia y la escalabilidad necesaria para evitar posibles errores durante el desarrollo de los servicios y su adaptación a la virtualización.

3. **Implementación sobre un convertidor real:** los resultados obtenidos en este trabajo han permitido la observación del correcto funcionamiento de un sistema completo con una gran diversidad de contextos *hardware* y *software*.

Los resultados experimentales obtenidos han permitido una validación exhaustiva del sistema implementado y la correcta interacción entre los subsistemas APU, RPU y PL sobre el MPSoC.

Capítulo 6

Conclusiones y líneas futuras

En este capítulo se resumen las principales conclusiones obtenidas y se proponen algunas posibles futuras líneas de investigación que se deriven del TFM desarrollado.

6.1 Conclusiones

En este trabajo se ha presentado un sistema de control y comunicaciones para la optimización del funcionamiento de las siguientes generaciones de micro-redes eléctricas, permitiendo el control de múltiples convertidores electrónicos de potencia, dispositivos que actúan de interfaces entre todos los elementos energéticos de la micro-red. La plataforma está basada en la combinación de la última generación de MPSoC, *Zynq UltraScale+* con la versatilidad de una FPGA, con una infraestructura que combina la interacción de distintos contextos de ejecución como son la APU, la RPU y la PL.

Además, se ha introducido el concepto de la virtualización como posible solución mixta entre una arquitectura de comunicaciones centralizada, a través del dominio *Dom0* maestro que permita comunicarse con las máquinas virtuales desplegadas y ofrecer el canal de comunicaciones con el usuario, y una arquitectura distribuida, donde cada VM se comunique con las demás y actúe en consecuencia.

Por otra parte, se han expuesto distintas alternativas para realizar la implementación del procesamiento asimétrico (AMP) entre *cores* mediante las librerías *OpenAMP*, con el protocolo *RPMsg* sobre el espacio de usuario y sobre el *kernel*. De igual forma, se ha ahondado en posibles arquitecturas de los servicios desarrollados sobre *Linux*, la alternativa monolítica y la basada en micro-servicios.

La propuesta presentada se ha analizado de forma exhaustiva tanto en términos de rendimiento y caracterización como sobre un convertidor real B2B de hasta 700 kW de potencia. Los resultados obtenidos han permitido validar el sistema completo, demostrando que la implementación de la comunicación *OpenAMP* en el *kernel* es mejor desde el punto de vista de fiabilidad en la transmisión de información (menor porcentaje de paquetes perdidos) y que la plataforma proporciona un medio adecuado para continuar con el desarrollo de infraestructuras que permitan ofrecer un mayor control y posibilidades en las futuras micro-redes.

6.2 Líneas futuras

En esta sección se presentan las posibles líneas de investigación del sistema de control y comunicaciones implementado. Las más relevantes se exponen a continuación:

- **Infraestructura completa basada en la virtualización:** con el objetivo fundamental de aprovechar las ventajas de aislamiento y despliegue de máquinas virtuales de las técnicas de virtualización, se pretende mejorar el sistema incorporando comunicación entre todos los *cores* A-53 de la APU, además de la ya incorporada con los núcleos de tiempo real R5 de la RPU.
- **Creación *device driver* junto con la virtualización:** como se ha demostrado la gestión de la comunicación *OpenAMP* ofrece resultados superiores sobre el *kernel*, por lo que sería interesante crear un *driver* compatible con el hipervisor *Xen*.
- **Control secundario:** en las máquinas virtuales desplegadas, incorporar algoritmos de regulación secundaria que permita mejorar la respuesta ante posibles perturbaciones severas sobre las microredes.
- ***Cloud Computing***: se pretende emplear la potencia de la Computación en la Nube para la implementación de controles avanzados con alto grado de disponibilidad y escalabilidad.

Capítulo 7

Pliego de condiciones

7.1 Introducción

En este apartado se evalúan las condiciones necesarias, tanto *hardware* como *software* para un correcto funcionamiento del sistema desarrollado en este TFM.

7.2 Requisitos *hardware*

- Procesador de 64 bits *multi-core*.
- Al menos 100GB de memoria libre en disco duro.
- 16GB de RAM DDR4 a 1867 MHz o superior.
- Kit de evaluación *Zynq UltraScale+* MPSoC ZCU102.

La plataforma de control y comunicaciones aplicada a las micro-redes eléctricas está basada en el empleo de la última generación de MPSoC *Zynq UltraScale+*, por lo que es necesario disponer de cualquier placa de desarrollo que integre este chip.

El conjunto de herramientas *PetaLinux* solo está disponible en el sistema operativo *Linux*, por lo que es necesario disponer de una máquina virtual en el ordenador. Dado que la compilación del núcleo de *Linux* es un proceso largo y costoso se recomienda disponer de un sistema multiprocesador que permita ejecutar la máquina virtual con soltura.

Así mismo, se recomienda disponer de al menos 100GB de disco duro libre para la instalación de los programas necesarios como *Vivado+SDK*, *Matlab*...etc, así como el despliegue de la máquina virtual junto con *Petalinux*.

7.3 Requisitos *software*

- Utilización de un sistema operativo *Windows* 10 de 64 bits.
- Utilización de una máquina virtual con el sistema operativo Ubuntu 16.04.
- Vivado+SDK 2018.2.

- Conjunto de herramientas de *Petalinux*.
- Matlab 2020a.
- Procesador de Latex.

En este apartado *software* se indican los sistemas operativos que se han utilizado para el desarrollo del sistema. La razoón fundamental de trabajar con Ubuntu 16.04, además de ser LTS, es por proporcionar la compatibilidad necesaria con la versión 2018.2 de *Xilinx* tanto de *Vivado* como de PetaLinux.

El empleo de *MatLab* se utiliza para realizar la decodificación del fichero binario de *logging* descargado a través del servicio web.

Capítulo 8

Presupuesto

8.1 Costes de equipamiento

8.1.1 Equipamiento hardware utilizado:

Concepto	Cantidad	Coste unitario	Subtotal(euros)
Ordenador i7 + 32GB de RAM	1	2500	2500
Kit de desarrollo MPSoC Zynq UltraScale+ ZCU102	1	2301,47	2301,47
Coste Total			4801,47

8.1.2 Recursos software utilizados:

Concepto	Cantidad	Coste unitario	Subtotal(euros)
Windows 10 Pro	1	259	259
Ubuntu 16.04 LTS	1	0	0
Xilinx Vivado Full+SDK+HLS 2018.2	1	3299	3299
Petalinux 2018.2	1	0	0
Matlab	1	2000	2000
Texstudio	1	0	0
Total			5558

8.2 Costes Mano de obra

Concepto	Cantidad	Coste unitario	Subtotal(euros)
Programacion y Desarrollo de Software	300	60euros/hora	18000
Mecanografiado de documentación, manuales y tutoriales	120	15 euros/hora	1800
Total			19800

8.3 Costes Totales

Concepto	Subtotal
Hardware	4801,47
Software	5558
Mano de obra	19800
Total	30159,47

El importe total del presupuesto asciende a la cantidad de: TREINTA MIL CIENTO CINCUENTA Y NUEVE EUROS

En Alcalá de Henares, a 1 de septiembre de 2021.

Sergio de López Diz Graduado en Máster Universitario en Ingeniería de Telecomunicaciones

Bibliografía

[1] *Global Renewable Outlook: Energy Transformation 2050.* IRENA, 2020. [Online]. Available: https://www.irena.org/publications/2020/Apr/Global-Renewables-Outlook-2020

[2] "Wide Bandgap Semiconductors (SiC/GaN)," disponible online: https://www.infineon.com/cms/en/product/wide-band-gap-semiconductors-sic-gan/ (Último acceso 06/04/2021).

[3] E. Espina, J. Llanos, C. Burgos-Mellado, R. Cardenas-Dobson, M. Martinez-Gomez, and D. Saez, "Distributed Control Strategies for Microgrids: An Overview," *IEEE Access*, vol. 8, pp. 193 412–193 448, oct 2020.

[4] A. Mohammed, S. S. Refaat, S. Bayhan, and H. Abu-Rub, "Ac microgrid control and management strategies: Evaluation and review," *IEEE Power Electronics Magazine*, vol. 6, no. 2, pp. 18–31, 2019.

[5] M. Savaghebi, A. Jalilian, J. C. Vasquez, and J. M. Guerrero, "Secondary control for voltage quality enhancement in microgrids," *IEEE Transactions on Smart Grid*, vol. 3, no. 4, pp. 1893–1902, 2012.

[6] R. Heydari, T. Dragicevic, and F. Blaabjerg, "High-bandwidth secondary voltage and frequency control of vsc-based ac microgrid," *IEEE Transactions on Power Electronics*, vol. 34, no. 11, pp. 11 320–11 331, 2019.

[7] F. Gao, R. Kang, J. Cao, and T. Yang, "Primary and secondary control in dc microgrids: a review," *Journal of Modern Power Systems and Clean Energy*, vol. 7, no. 2, pp. 227–242, 2019.

[8] X. Lu, J. M. Guerrero, K. Sun, and J. C. Vasquez, "An improved droop control method for dc microgrids based on low bandwidth communication with dc bus voltage restoration and enhanced current sharing accuracy," *IEEE Transactions on Power Electronics*, vol. 29, no. 4, pp. 1800–1812, 2014.

[9] Y. Zhang, X. Yuan, X. Wu, Y. Yuan, and J. Zhou, "Parallel implementation of model predictive control for multilevel cascaded h-bridge statcom with linear complexity," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 2, pp. 832–841, 2020.

[10] L. He, F. Wang, J. Wang, and J. Rodríguez, "Zynq implemented luenberger disturbance observer based predictive control scheme for pmsm drives," *IEEE Transactions on Power Electronics*, vol. 35, no. 2, pp. 1770–1778, 2020.

[11] N. Mohan, T. M. Undeland, and W. P. Robbins, *Power Electronics. Converters, Applications and Design*, 3rd ed. John Wiley and Sons, Inc, 2003.

[12] "Zynq ultrascale+ device," *Technical Reference Manual.* [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf

[13] Xilinx, "Managing power and performance with the zynq ultrascale+ mpsoc," 2016. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp482-zu-pwr-perf.pdf

[14] L. Crockett, R. Elliot, M. A. Enderwitz, and R. Stewart, "The zynq book: Embedded processing with the arm cortex-a9 on the xilinx zynq-7000 all programmable soc," 2014.

[15] guru99, "Get vs post: Key difference between http methods." [Online]. Available: https://www.guru99.com/difference-get-post-http.html

[16] M. Organization, "Modbus messaging on tcp/ip implementation guide v1.0b," 2006. [Online]. Available: https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf

[17] "Overview xen project." [Online]. Available: https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

[18] M. Jarnut, S. Wermi?ski, and B. Wa?kowicz, "Comparative analysis of selected energy storage technologies for prosumer-owned microgrids," *Renewable and Sustainable Energy Reviews*, vol. 74, pp. 925 – 937, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032117303143

[19] Y. Amanbek, Y. Tabarak, H. S. V. S. K. Nunna, and S. Doolla, "Decentralized transactive energy management system for distribution systems with prosumer microgrids," in *2018 19th International Carpathian Control Conference (ICCC)*, 2018, pp. 553–558.

[20] J. C. Balda and A. Mantooth, "Power-semiconductor devices and components for new power converter developments: A key enabler for ultrahigh efficiency power electronics," *IEEE Power Electronics Magazine*, vol. 3, no. 2, pp. 53–56, 2016.

[21] A. Hirsch, Y. Parag, and J. Guerrero, "Microgrids: A review of technologies, key drivers, and outstanding issues," *Renewable and Sustainable Energy Reviews*, vol. 90, pp. 402–411, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S136403211830128X

[22] D. Newman, "Right-Sizing the Grid," *Mechanical Engineering*, vol. 137, no. 01, pp. 34–39, 01 2015. [Online]. Available: https://doi.org/10.1115/1.2015-Jan-2

[23] "Enabling virtualization with xen hypervisor on zynq ultrascale+ mpsocs white paper (wp474)," 2016.

[24] Y. Khayat, Q. Shafiee, R. Heydari, M. Naderi, T. Dragi?evi?, J. W. Simpson-Porco, F. Dörfler, M. Fathi, F. Blaabjerg, J. M. Guerrero, and H. Bevrani, "On the secondary control architectures of ac microgrids: An overview," *IEEE Transactions on Power Electronics*, vol. 35, no. 6, pp. 6482–6500, 2020.

[25] A. Bidram, V. Nasirian, A. Davoudi, and F. Lewis, *Cooperative Synchronization in Distributed Microgrid Control*, 01 2017.

[26] H. Cha, T.-K. Vu, and J.-E. Kim, "Design and control of proportional-resonant controller based photovoltaic power conditioning system," in *2009 IEEE Energy Conversion Congress and Exposition*, 2009, pp. 2198–2205.

[27] J. Hu, J. Zhu, and D. G. Dorrell, "Model predictive control of grid-connected inverters for pv systems with flexible power regulation and switching frequency reduction," *IEEE Transactions on Industry Applications*, vol. 51, no. 1, pp. 587–594, 2015.

[28] E. Rokrok and M. E. Hamedani Golshan, "Adaptive voltage droop scheme for voltage source converters in an islanded multibus microgrid," *IET Generation Transmission & Distribution*, vol. 4, p. 562?578, 06 2010.

[29] W. Yao, M. Chen, J. Matas, J. M. Guerrero, and Z.-M. Qian, "Design and analysis of the droop control method for parallel inverters considering the impact of the complex impedance on the power sharing," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 2, pp. 576–588, 2011.

[30] D. E. Olivares, A. Mehrizi-Sani, A. H. Etemadi, C. A. Cañizares, R. Iravani, M. Kazerani, A. H. Hajimiragha, O. Gomis-Bellmunt, M. Saeedifard, R. Palma-Behnke, G. A. Jiménez-Estévez, and N. D. Hatziargyriou, "Trends in microgrid control," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1905–1919, 2014.

[31] J. W. Simpson-Porco, Q. Shafiee, F. Dörfler, J. C. Vasquez, J. M. Guerrero, and F. Bullo, "Secondary frequency and voltage control of islanded microgrids via distributed averaging," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 11, pp. 7025–7038, 2015.

[32] M. Andishgar, E. Gholipour, and R.-A. Hooshmand, "Voltage quality enhancement in islanded microgrids with multi voltage quality requirements at different buses," *IET Generation, Transmission & Distribution*, vol. 12, 02 2018.

[33] A. Micallef, M. Apap, C. Spiteri-Staines, J. M. Guerrero, and J. C. Vasquez, "Reactive power sharing and voltage harmonic distortion compensation of droop controlled single phase islanded microgrids," *IEEE Transactions on Smart Grid*, vol. 5, no. 3, pp. 1149–1158, 2014.

[34] A. Milczarek, M. Malinowski, and J. M. Guerrero, "Reactive power management in islanded microgrid proportional power sharing in hierarchical droop control," *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1631–1638, 2015.

[35] M. Jafari, V. Sarfi, A. Ghasemkhani, H. Livani, L. Yang, H. Xu, and R. Koosha, "Adaptive neural network based intelligent secondary control for microgrids," in *2018 IEEE Texas Power and Energy Conference (TPEC)*, 2018, pp. 1–6.

[36] S. Riverso, M. Tucci, J. C. Vasquez, J. M. Guerrero, and G. Ferrari-Trecate, "Stabilizing plug-and-play regulators and secondary coordinated control for ac islanded microgrids with bus-connected topology," *Applied Energy*, vol. 210, pp. 914–924, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261917311303

[37] G. Lou, W. Gu, W. Sheng, X. Song, and F. Gao, "Distributed model predictive secondary voltage control of islanded microgrids with feedback linearization," *IEEE Access*, vol. 6, pp. 50 169–50 178, 2018.

[38] H. Cai and G. Hu, "Distributed nonlinear hierarchical control of ac microgrid via unreliable communication," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2429–2441, 2018.

[39] M. S. Golsorkhi, Q. Shafiee, D. D.-C. Lu, and J. M. Guerrero, "A distributed control framework for integrated photovoltaic-battery-based islanded microgrids," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2837–2848, 2017.

[40] W. Kang, Q. Li, M. Gao, X. Li, J. Wang, R. Xu, and M. Chen, "Distributed secondary control method for islanded microgrids with communication constraints," *IEEE Access*, vol. 6, pp. 5812–5821, 2018.

[41] W. P. M. H. Heemels, M. C. F. Donkers, and A. R. Teel, "Periodic event-triggered control for linear systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 847–861, 2013.

[42] Y. Han, H. Li, L. Xu, X. Zhao, and J. M. Guerrero, "Analysis of washout filter-based power sharing strategy?an equivalent secondary controller for islanded microgrid without lbc lines," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 4061–4076, 2018.

[43] E. Weitenberg, Y. Jiang, C. Zhao, E. Mallada, C. De Persis, and F. Dörfler, "Robust decentralized secondary frequency control in power systems: Merits and tradeoffs," *IEEE Transactions on Automatic Control*, vol. 64, no. 10, pp. 3967–3982, 2019.

[44] J. Lu, M. Savaghebi, and J. M. Guerrero, "Second order washout filter based power sharing strategy for uninterruptible power supply," in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017, pp. 7854–7859.

[45] Y. Khayat, M. Naderi, Q. Shafiee, M. Fathi, H. Bevrani, T. Dragicevic, and F. Blaabjerg, "Communication-less optimal frequency control of islanded microgrids," in *2018 20th European Conference on Power Electronics and Applications (EPE'18 ECCE Europe)*, 2018, pp. P.1–P.8.

[46] Y. Khayat, M. Naderi, Q. Shafiee, Y. Batmani, M. Fathi, J. M. Guerrero, and H. Bevrani, "Decentralized optimal frequency control in autonomous microgrids," *IEEE Transactions on Power Systems*, vol. 34, no. 3, pp. 2345–2353, 2019.

[47] W. Gu, G. Lou, W. Tan, and X. Yuan, "A nonlinear state estimator-based decentralized secondary voltage control scheme for autonomous microgrids," *IEEE Transactions on Power Systems*, vol. 32, no. 6, pp. 4794–4804, 2017.

[48] J. Liu, W. Zhang, and G. Rizzoni, "Robust stability analysis of dc microgrids with constant power loads," *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 851–860, 2018.

[49] J. Beerten and R. Belmans, "Analysis of power sharing and voltage deviations in droop-controlled dc grids," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 4588–4597, 2013.

[50] G. Xu, D. Sha, and X. Liao, "Decentralized inverse-droop control for input-series?output-parallel dc?dc converters," *IEEE Transactions on Power Electronics*, vol. 30, no. 9, pp. 4621–4625, 2015.

[51] F. Chen, R. Burgos, D. Boroyevich, and W. Zhang, "A nonlinear droop method to improve voltage regulation and load sharing in dc systems," in *2015 IEEE First International Conference on DC Microgrids (ICDCM)*, 2015, pp. 45–50.

[52] H. Kakigano, Y. Miura, and T. Ise, "Distribution voltage control for dc microgrids using fuzzy control and gain-scheduling technique," *IEEE Transactions on Power Electronics*, vol. 28, no. 5, pp. 2246–2258, 2013.

[53] T. Dragicevic, J. C. Vasquez, J. M. Guerrero, and D. Skrlec, "Advanced lvdc electrical power architectures and microgrids: A step toward a new generation of power distribution networks." *IEEE Electrification Magazine*, vol. 2, no. 1, pp. 54–65, 2014.

[54] D. Wu, F. Tang, T. Dragicevic, J. M. Guerrero, and J. C. Vasquez, "Coordinated control based on bus-signaling and virtual inertia for islanded dc microgrids," *IEEE Transactions on Smart Grid*, vol. 6, no. 6, pp. 2627–2638, 2015.

[55] T. R. Oliveira, W. W. A. Gonçalves Silva, and P. F. Donoso-Garcia, "Distributed secondary level control for energy storage management in dc microgrids," *IEEE Transactions on Smart Grid*, vol. 8, no. 6, pp. 2597–2607, 2017.

[56] J. Schonbergerschonberger, R. Duke, and S. Round, "Dc-bus signaling: A distributed control strategy for a hybrid renewable nanogrid," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 5, pp. 1453–1460, 2006.

[57] K. Sun, L. Zhang, Y. Xing, and J. M. Guerrero, "A distributed control strategy based on dc bus signaling for modular photovoltaic generation systems with battery energy storage," *IEEE Transactions on Power Electronics*, vol. 26, no. 10, pp. 3032–3045, 2011.

[58] L. Meng, T. Dragicevic, J. Roldán-Pérez, J. C. Vasquez, and J. M. Guerrero, "Modeling and sensitivity study of consensus algorithm-based distributed hierarchical control for dc microgrids," *IEEE Transactions on Smart Grid*, vol. 7, no. 3, pp. 1504–1515, 2016.

[59] T. Morstyn, B. Hredzak, and V. G. Agelidis, "Cooperative multi-agent control of heterogeneous storage devices distributed in a dc microgrid," *IEEE Transactions on Power Systems*, vol. 31, no. 4, pp. 2974–2986, 2016.

[60] A. Tah and D. Das, "An enhanced droop control method for accurate load sharing and voltage improvement of isolated and interconnected dc microgrids," *IEEE Transactions on Sustainable Energy*, vol. 7, no. 3, pp. 1194–1204, 2016.

[61] S. Peyghami, H. Mokhtari, P. C. Loh, P. Davari, and F. Blaabjerg, "Distributed primary and secondary power sharing in a droop-controlled lvdc microgrid with merged ac and dc characteristics," *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 2284–2294, 2018.

[62] A. Bani-Ahmed, L. Weber, A. Nasiri, and H. Hosseini, "Microgrid communications: State of the art and future trends," in *2014 International Conference on Renewable Energy Research and Application (ICRERA)*, 2014, pp. 780–785.

[63] E. Ancillotti, R. Bruno, and M. Conti, "The role of communication systems in smart grids: Architectures, technical solutions and research challenges," *Computer Communications*, vol. 36, no. 17, pp. 1665–1697, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366413002090

[64] A. Kantamneni, L. E. Brown, G. Parker, and W. W. Weaver, "Survey of multi-agent systems for microgrid control," *Engineering Applications of Artificial Intelligence*, vol. 45, pp. 192–203, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197615001529

[65] S. Marzal, R. Salas, R. González-Medina, G. Garcerá, and E. Figueres, "Current challenges and future trends in the field of communication architectures for microgrids," *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 3610–3622, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032117314703

[66] K. Venkatraman, B. Dastagiri Reddy, M. Selvan, S. Moorthi, N. Kumaresan, and N. A. Gounden, "Online condition monitoring and power management system for standalone micro-grid using fpgas," *IET Generation, Transmission & Distribution*, vol. 10, no. 15, pp. 3875–3884, 2016. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2016.0445

[67] D. Di Mauro, J. C. Augusto, A. Origlia, and F. Cutugno, "A framework for distributed interaction in intelligent environments," in *Ambient Intelligence*, A. Braun, R. Wichert, and A. Maña, Eds. Cham: Springer International Publishing, 2017, pp. 136–151.

[68] S. Howell, Y. Rezgui, J.-L. Hippolyte, B. Jayan, and H. Li, "Towards the next generation of smart grids: Semantic and holonic multi-agent management of distributed energy resources," *Renewable and Sustainable Energy Reviews*, vol. 77, pp. 193–214, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032117304392

[69] S. Wendel, A. Geiger, E. Liegmann, D. Arancibia, E. Durán, T. Kreppel, F. Rojas, F. Popp-Nowak, M. Diaz, A. Dietz, R. Kennel, and B. Wagner, "Ultrazohm - a powerful real-time computation platform for mpc and multi-level inverters," in *2019 IEEE International Symposium on Predictive Control of Electrical Drives and Power Electronics (PRECEDE)*, 2019, pp. 1–6.

[70] A. Nabae, I. Takahashi, and H. Akagi, "A new neutral-point-clamped pwm inverter," *IEEE Transactions on Industry Applications*, vol. IA-17, no. 5, pp. 518–523, 1981.

[71] T. E. Michael Frisch, "Advantages of npc inverter topologies with power modules," in *Vincotech*, 2009.

[72] H. Beck and R. Hesse, "Virtual synchronous machine," *2007 9th International Conference on Electrical Power Quality and Utilisation*, pp. 1–6, 2007.

[73] B. B. Johnson, M. Sinha, N. G. Ainsworth, F. Dörfler, and S. V. Dhople, "Synthesizing virtual oscillators to control islanded inverters," *IEEE Transactions on Power Electronics*, vol. 31, no. 8, pp. 6002–6015, 2016.

[74] "Zynq ultrascale+ mpsoc datasheet." [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[75] "Libmetal and openamp user guide (ug1186). xilinx." [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1186-zynq-openamp-gsg.pdf

[76] M. G. Corporation, "Openamp framework user reference," 2010-2014.

[77] H. Nielsen, J. Mogul, L. M. Masinter, R. T. Fielding, J. Gettys, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, Jun. 1999. [Online]. Available: https://rfc-editor.org/rfc/rfc2616.txt

[78] D. Dube and J. Camerini, "MODBUS Application Protocol," Internet Engineering Task Force, Internet-Draft draft-dube-modbus-applproto-00, May 2002, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-dube-modbus-applproto-00

[79] A. Aguiar and F. Hessel, "Embedded systems' virtualization: The next challenge?" 07 2010, pp. 1 – 7.

[80] R. M. López, "Mejoras de control de sistemas electricos con integración masiva de convertidores ante perturbaciones severas," Trabajo Fin de Máster, Escuela Politécnica Superior, 2021.

[81] G. C. Goodwin, S. F. Graebe, and M. E. Salgado, *Control System Design*, 1st ed. USA: Prentice Hall PTR, 2000.

Apéndice A

Manual de usuario

A.1 Introducción

Este manual de usuario especifica la funcionalidad de la plataforma de comunicaciones y control desarrollada en este TFM así como una descripción del manejo de la misma a través del servicio web.

A.2 Requisitos previos

El sistema de control y comunicaciones aplicado a los convertidores pertenecientes a una micro-red está basado en el empleo de un MPSoC, donde se distribuye la funcionalidad entre los distintos núcleos disponibles. El correcto funcionamiento de la plataforma descrita en este documento supone cumplir los siguientes requisitos:

- **Conexión con la placa de desarrollo.** Para que el usuario pueda controlar los distintos elementos de la micro-red es necesario disponer de una conexión *Ethernet* directa con el MPSoC. La placa de control puede disponer de acceso a Internet para extender su acceso a un lugar remoto (aunque no es necesario para su correcto funcionamiento).
- **Ordenador con un navegador web.** Dado que el sistema dispone de un servidor web que sirve el contenido, es necesario disponer de una estación de trabajo que disponga de un navegador que permita acceder al servicio web implementado.
- **MatLab.** Permite de forma sencilla decodificar el archivo de *logging* descargado a través de la página web para el análisis gráfico de los resultados experimentales obtenidos.

A.3 Descripción general

El acceso al panel de control principal se realiza a través de una página de inicio de sesión en la que se debe introducir un usuario y una contraseña, tal y como se muestra en la figura A.1.

Si las credenciales introducidas son correctas, el usuario será redirigido directamente a la página de control principal que se adjunta en la figura A.2.

En ella, se observa con claridad un esquema unifilar del equipo diseñado, donde se dispone de un transformador de entrada (T1) que permite la conexión del VSC1 con la red a través de un filtro L. Por

Figura A.1: Página web de inicio de sesión.

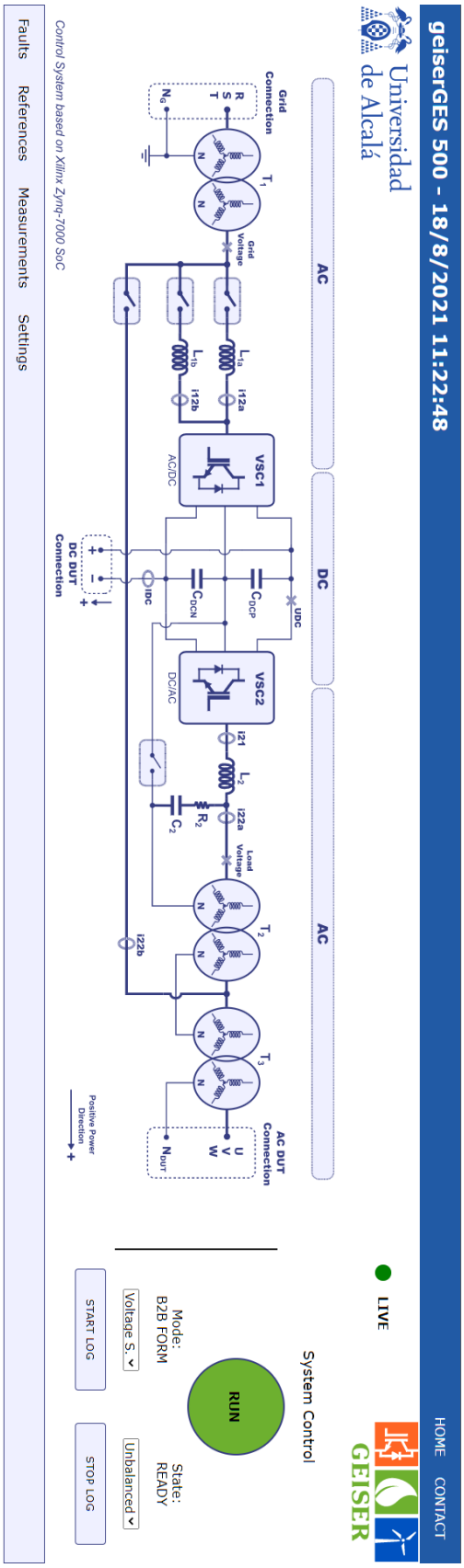


Figura A.2: Página web de control principal.

otro lado, en el centro de la imagen se vislumbra el bus de continua con acceso al punto medio a través del cuál se realiza la conexión con la parte DC del DUT con el que se esté trabajando.

En la parte del VSC2, se muestra el filtro de salida LC así como los transformadores T2 y T3 que permiten la conexión a la red y a la parte de alterna del DUT respectivamente.

A.3.1 Panel de control

Junto al esquema del equipo, se dispone de un panel de control que dispone de selectores para elegir el modo de funcionamiento y de control, así como un único botón que establece la secuencia para arrancar el equipo.

Los modos de operación se listan a continuación:

1. ***VSC1***: permite trabajar como rectificador activo, controlando la tensión del bus de continua así como intercambiar potencia reactiva con la red.
2. ***Voltage Source***: permite trabajar como una fuente de tensión regulable. Se posibilita la modificación del valor de tensión RMS, la frecuencia y el ángulo de las 3 fases para la realización de pruebas de huecos.

Los estados por los que transcurre el convertidor en el modo de operación VSC1 son los siguientes:

- ***UDC CHARGE***: carga suave de los condensadores del bus DC desde el lado de red.
- ***VSC1 INIT***: el bus DC se encuentra cargado esperando comenzar a modular en el lado VSC1 para controlar la tensión del bus y la potencia reactiva.
- ***VSC1 ON***: se está modulando en el VSC1, controlando el bus DC y la potencia reactiva.

Los estados pertenecientes al modo *Voltage Source* son los descritos anteriormente más los siguientes:

- ***B2B INIT***: con el VSC1 modulando, se espera hasta que el control de VSC1 se encuentre estable para comenzar el lado VSC2.
- ***B2B ON***: se modula y controla simultáneamente en VSC1 y VSC2.

Los estados principales del convertidor que no pertenecen a ningún modo de funcionamiento son los siguientes:

- ***READY***: el sistema está preparado para comenzar a controlar.
- ***OFF***: todas las PWM están detenidas y los contactores abiertos.
- ***RESET***: se realiza la reconfiguración del equipo para volver al estado *READY*.

Además, en esta sección se incluyen dos botones dedicados a la descarga del fichero de *logging*. Si el usuario presiona el botón “START LOG”, comienza de forma automática la descarga de un fichero binario que contiene información acerca de las señales adquiridas por el equipo así como variables internas del sistema. Para detener la descarga , se debe pulsar el botón “STOP LOG”. En la figura A.3 se muestra en detalle el panel de control del sistema junto con todos sus elementos.

Sobre la página principal se permite navegar por diferentes pestañas, las cuáles se describen en los siguientes subapartados.

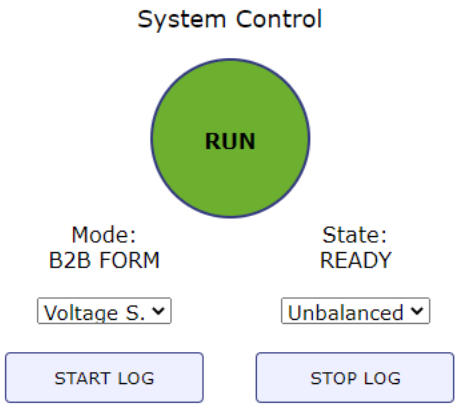


Figura A.3: Panel del control del sistema.

A.3.2 Panel de fallos

En el panel de fallos (figura A.4) se incluye la visualización de unos leds asociados a cada uno de los posibles fallos definidos en el diseño del equipo, entre los que se incluyen:

- Fallos en los módulos de IGBTs del VSC1.
- Fallos en los módulos de IGBTs del VSC2.
- Sobrepaso del límite configurado en la tensión del bus DC.
- Sobrepaso de los límites de tensión en las fases de red.
- Sobrepaso de los límites de tensión en las fases de salida del VSC2.
- Sobrepaso de límites de las corrientes.
- Sobrepaso de temperatura en los IGBTs.
- Notificación de la fuente de detección del error: *hardware* o *software*.

Además, se incorpora un código de error en hexadecimal que resume todas las fuentes de error que se han producido durante el ensayo. La notificación de un fallo en una determinada señal consiste en el paso de color verde a color rojo del correspondiente led asociado.

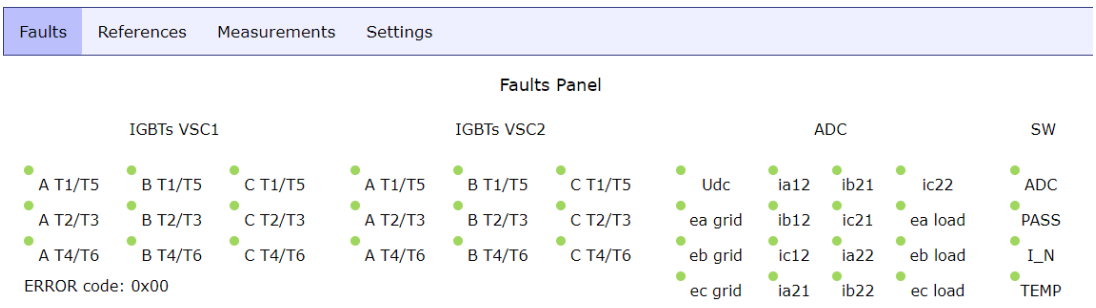


Figura A.4: Detalle panel de fallos.

A.3.3 Panel de referencias

El panel de referencias (figura A.5) permite al usuario la introducción de consignas para múltiples señales. Se diferencian dos secciones: “INMEDIATE” y “PROGRAMMABLE”. La primera de ellas permite la modificación de la referencia de una determinada señal de forma inmediata, es decir, en forma de escalón. Por otro lado, la segunda opción flexibiliza el sistema permitiendo la realización de secuencias. El formato correcto para introducir las secuencias consiste en introducir en el campo “VALUES” las referencias separadas por espacios y entre corchetes (p.ej: [1 2 3 4]). De forma similar, se debe ingresar un vector con el mismo número de elementos en el campo “TIME”, además de ser creciente (p.ej:[1 1.001 2 4]). Los valores introducidos en el vector de tiempo deben ser segundos.

Una vez insertados los vectores, el usuario debe pulsar el botón “SET PROGRAMMABLE” para almacenar la secuencia en el servidor (las secuencias configuradas se reflejan de forma visual en una gráfica para que el usuario compruebe que la secuencia introducida es correcta). Para comenzar las secuencias se debe pulsar el botón “START”. Para limpiar el registro de secuencias se debe pulsar el botón “CLEAR”.

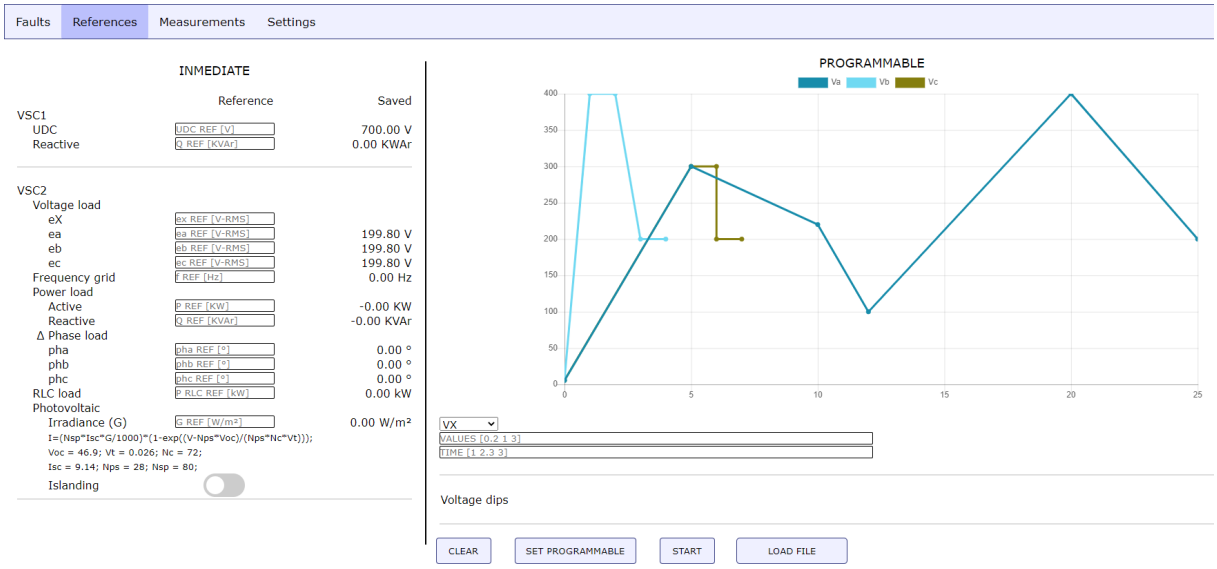


Figura A.5: Detalle panel de referencias.

Una funcionalidad adicional es la posibilidad de subir un fichero CSV para la configuración automática de las secuencias. Se puede crear en *Excel* un fichero CSV donde se introduzcan los vectores de “valores” y “tiempo” de múltiples señales con el formato que se muestra en la figura A.6. Esto flexibiliza aún más la plataforma, permitiendo al cliente guardar los ensayos en ficheros CSV para reproducirlos de forma más rápida en el futuro.

	A	B	C	D	E	F
1	Va	Time	Vb	Time2	Vc	Time3
2	100	2	50	2	50	1
3	100	3	100	3	50	6
4	200	5	150	5	100	8
5	230	10	100	10	200	11

Figura A.6: Ejemplo creación fichero CSV en Excel.

En este caso, el usuario debe pulsar el botón “LOAD FILE” para subir el fichero CSV para almacenar en el servidor los vectores configurados y, nuevamente, pulsar el botón “START” cuando desee iniciar las secuencias.

A.3.4 Panel de medidas

En el panel de medidas (figura A.7) se resumen los valores de las señales adquiridas con su correspondientes referencias. Se encuentra dividida en dos secciones: VSC1 y VSC2. Además, se incluye información acerca de la transferencia de potencia a ambos lados del convertidor.

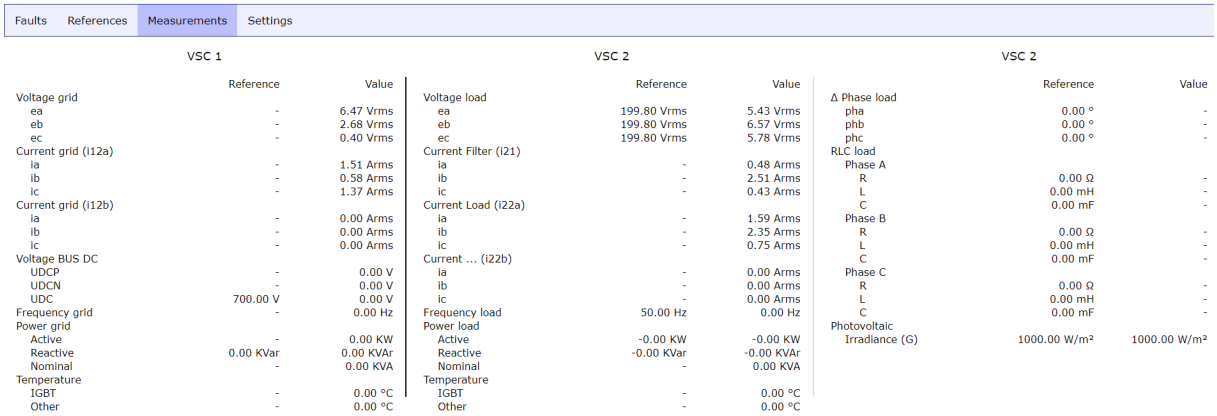


Figura A.7: Detalle panel de medidas.

A.3.5 Panel de configuración

El panel de configuración (figura A.8) es únicamente accesible cuando el equipo se encuentra en el estado "READY", es decir, preparado para comenzar a controlar el convertidor. Posibilita la modificación de forma dinámica de la frecuencia de conmutación de los IGBTs, la frecuencia de control y la de comunicación OpenAMP. Además, permite configurar los límites asociados a las señales que se listan a continuación:

- Límite de tensiones de red y tensiones de carga.
- Límite corrientes entrantes al VSC1 (i12a e i12b) y salientes al VSC2 (i21, i22a e i22b).
- Límite tensión máxima y desbalanceo bus DC.

Si el usuario pulsa el botón "DEFAULT", los límites configurados son aquellos definidos en la aplicación *baremetal* como los más adecuados por diseño.

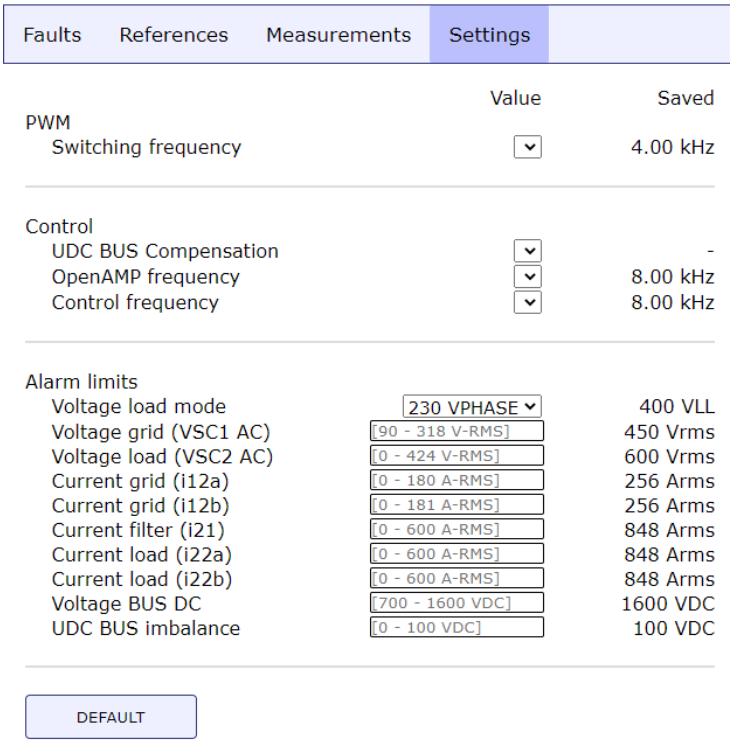


Figura A.8: Detalle panel de configuración.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá